Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros

TFM del Máster Inteligencia de Negocio y Big Data en Entornos Seguros

Construcción de un sistema POS utilizando Reconocimiento de Objetos con Deep Learning

Presentado por Thi Lan Phuong Dinh en Universidad de Valladolid - 18 de febrero de 2025

Tutor: Dr. Quiliano Isaac Moro Sancho

Resumen

La idea de este proyecto es construir un sistema POS (Sistema de Punto de Venta) usando Reconocimiento de Objetos con Deep Learning.

El sistema POS tiene dos tipos: POS F&B (POS Alimentos y Bebidas) y POS Retails (Punto de Venta para Comercios Minoristas). En este trabajo, uno de los objetivos será poder construir una prueba de concepto o demostrador del POS F&B, orientado a reconocer platos preparados a partir de las fotografías. También se incluiría un recomendador de platos similares disponibles en la carta del restaurante.

El sistema POS no sólo permite pedir comida desde los dispositivos móviles de los clientes, como teléfonos o tabletas, sino que también les permite escanear fotografías de la comida en la carta del restaurante o bar. El núcleo del sistema es el reconocimiento de objetos con aprendizaje automático para reconocer el artículo y, si es necesario (por ejemplo, porque ya no está disponible), recomendar opciones similares.

Tras identificar el plato y una vez que el cliente termine de elegir, el sistema enviará el pedido a cocina y los chefs prepararán el plato.

Descriptores

Big Data, Photographs, Business Intelligence, Python, YOLOv8, YOLOv11, Labelme, Labelme2YOLO, JavaScript, Google Colab, Similarity, HTML, Adobe Illustrator, InVision, UXPin.

Abstract

The idea of this project is to build a POS system (Point of Sale system) using Object Recognition with Deep Learning.

The POS system has two types: POS F&B (POS Food and Beverage) and POS Retails. In this work, one of the objectives will be able to build a proof of concept or demonstrator of the POS F&B, aimed at recognizing dishes prepared from the photos. A recommender of similar dishes available on the restaurant's menu would also be included.

The POS system is not only order food from customers' mobile devices such as phones or tablets, but also allow them scan photos of the food in the menu of the restaurant or bar. The core of the system is object recognition with deep learning to recognize the item and, if necessary (for example, because it is no longer available), recommend similar options.

After identifying the dish and once customer finish choosing, the system will send the order to the kitchen and the chefs will prepare the dish.

Keywords

Big Data, Photographs, Business Intelligence, Python, YOLOv8, YOLOv11, Labelme, Labelme2YOLO, JavaScript, Google Colab, Similarity, HTML, Adobe Illustrator, InVision, UXPin.

Índice general

Índice general III
Índice de figuras V
Índice de tablas VI

Memoria 1

1. Introducción 2

2. Objetivos del proyecto 4

3. Conceptos teóricos 9
3.1. Sistema POS F&B 9
3.2. YOLO - Real Time Object Detection 14
3.3. Integración con el sistema POS 20

4. Técnicas y herramientas 22
4.1. Metodología 22
4.2. Herramientas y Tecnologías 26

5. Aspectos relevantes del desarrollo del proyecto 33
5.1. Deseño y Prototyping 33
5.2. Colección de datos 52
5.3. Entrenamiento del modelo 61
5.4. POS de demostración y recomendador 70
5.5. Pruebas 93

6. Conclusiones y Líneas de trabajo futuras 95

Apéndices **100**

Apéndice A Plan de Proyecto Software 101
A.1. Introducción 101
A.2. Planificación temporal 101
A.3. Estudio de viabilidad 103
Apéndice B Especificación de Requisitos 106
B.1. Introducción 106
B.2. Objetivos generales 106
B.3. Catalogo y especificación de requisitos 107
Apéndice C Documentación técnica de programación 109
D.1. Introducción 109
D.2. Estructura de directorios 109
D.3. Compilación, instalación y ejecución del proyecto 111

Bibliografía 114

Índice de figuras

3.1. Arquitectura de YOLOv8 presentada por el usuario RangeKing 15
3.2. Aumento de Mosaic de fotografías de tableros de ajedrez–Roboflow 16
3.3. Visualización de una caja de anclaje en YOLO – Roboflow 16
3.4. Nuevo módulo YOLOv8 C2f 17
3.5. YOLOv8 Decoupled Head - MMLab 17
4.1. Change runtime type en Goole Colab 24
5.1. Proceso de pedido en la versión de la aplicación móvil para clientes 34
5.2. Proceso de pedido en la versión de la aplicación para el personal 34
5.3. Pantalla de registro de cliente 35
5.4. Eliminar una cuenta 35
5.5. Pestaña de menú después de seleccionar una mesa 36
5.6. Ingredientes del plato con advertencia alérgica 36
5.7. Confirmar un pedido 37
5.8. Lista de pedidos con factura temporal 37
5.9. Cancelar un pedido 38
5.10. Pedir por voz 39
5.11. Realiza el pedido escaneando una imagen 39
5.12. Identificación de pinchos en tiempo real 39
5.13. Realiza el pedido subiendo una imagen 40
5.14. Recomendación de pinchos similares 40
5.15. Botón de llamada al camarero 41
5.16. Estado del pedido 41
5.17. Revisión de factura y elección de método de pago 42
5.18. Menú de configuración del cliente 42
5.19. Historial de pedidos del cliente 43
5.20. Gestión de mesa 44
5.21. Pestaña Menú en el sistema POS – versión para personal 44
5.22. Agregar cliente a una mesa abierta 45

5.23. Pantalla de cocina con el estado de los platos 45
5.24. Pantalla de menú en la aplicación POS - versión para personal 46
5.25. Menú de configuración del personal 46
5.26. Inventario 47
5.27. Gestión de clientes 47
5.28. Un ejemplo de informe de ventas 49
5.29. Pantalla de cocina – Visualización por mesa 50
5.30. Pantalla de cocina – Visualización por plato 50
5.31. Etiqueta una imagen en labelme 55
5.32. Etiquetas de polígonos por ID de grupo en labelme 56
5.33. Un archivo en formato JSON después del etiquetado 56
5.34. Otros ejemplos de etiquetas de imágenes usando LabelMe y conversión de anotaciones al formato JSON 57
5.35. Directorio de conjunto de datos en Google Drive 61
5.36. Entrenamiento del modelo YOLOv8 en Google Colab 63
5.37. Resultados obtenidos tras el entrenamiento con YOLOv8 63
5.38. Visualización de lotes de entrenamiento de YOLOv8 64
5.39. Visualización de lotes de validación de YOLOv8 65
5.40. Gráfico normalizado de matriz de confusión 65
5.41. Gráfico de correlograma de etiquetas 66
5.42. Gráfico de distribución de etiquetas 67
5.43. Gráfico de curva de confianza F1 68
5.44. Predicción de pinchos 69
5.45. Página de inicio de POS de demostración 76
5.46. Página de inicio de login de POS de demostración 76
5.47. Página de registro 76
5.48. Página de perfil de usuario 77
5.49. Página de menú 77
5.50. Página de confirmación del pedido 77
5.51. Página de inventario 78
5.52. Página de la pantalla de cocina 78
5.53. Página de Dashboard – Ingresos por ventas diarias 79
5.54. Página de Dashboard - Los 5 pinchos más vendidos 79
5.55. Puntuación de similitud entre los ingredientes de los pichos 90
5.56. Resultado de una recomendación de artículo similar 91
5.57. Declaraciones del registro de depuración (debug statements) 94
6.1. Revisión y votación del cliente 98
A.1. TFM Milestone 102
A.2. Agile Board TFM 103
C.1. Iniciación la aplicación de demostración 112

Índice de tablas

3.1. Evaluación YOLOv8 COCO – Ultralytics 18
3.2. Herramientas y tecnologías utilizadas en cada parte del proyecto 26
5.1. Prototype 51
5.2. Lista de imágenes de los pinchos correspondientes a cada número ... 55
5.3. Dimensiones de datos de pincho 71
5.4. Dimensiones de datos de la mesa 71
5.5. Dimensiones de datos del pedido 71
5.6. Dimensiones de datos del pedido por cada mesa 71
5.7. Dimensiones de datos del usuasrio 71
5.8. Historial de los pedidos completados o pasados 71
5.9. Registro de pedidos de cocina 71
5.10. Estructura de la interfaz de usuario 75
A.1. Presupuesto del proyecto 104
A.2. Licencias utilizadas para desarrollar el proyecto 105
B.1. Requisito 1: Evaluación inicial del proyecto 107
B.2. Requisito 2 - Operar el sistema de detección de objetos 107
B.3. Requisito 3: Inicialice el entorno de desarrollo en Google Colab conectándose al conjunto de datos en Google Drive 107

B.4. Funcionalidad 1: Detección y predicción de objetos utilizando best.pt del modelo de entrenamiento YOLOv8 107
B.5. Funcionalidad 2: procesamiento de pedidos utilizando Flask (web framework) 108
B.6. Funcionalidad 3 – Actualización de inventario 108
B.7. Funcionalidad 4 – Recomendador de artículos similares 108
B.8. Funcionalidad 5: interfaz de usuario (una demostración de POS en una aplicación web que se ejecuta en el entorno Python) 108

En primer lugar, quiero agradecer sinceramente a los profesores de la Universidad de Valladolid, de la Universidad de Burgos y de la Universidad de León por facilitarme conocimientos relacionados con este Máster.
Gracias a los profesores de materias en el campo de Business Intelligence como Conceptos Financieros y Herramientas de Gestión en las Empresas, Procesamiento de Datos para Business Intelligence, Business Intelligence Aplicada (I y II) y Visualización de Datos por no solo aportar un conocimiento más amplio en este campo, también me inspiró a realizar este proyecto para el TFM.
Y sobre todo quiero agradecer al Dr. Quiliano Isaac Moro Sancho por acompañarme y guiarme para completar el proyecto. Estoy muy agradecida por su ayuda y su entusiasmo.

Memoria

Introducción

Hoy en día la tecnología avanza a un ritmo vertiginoso, si nos dormimos sólo unos días, parece que nos hemos perdido una década entera. “Dominar la tecnología es dominar el futuro”, eso es algo que todo el mundo sabe, pero no todo el mundo puede hacer.

La misión de la “gente de TI” como nosotros es llevar la tecnología a todos, en un mundo perfecto, cualquier usuario podría usarla. En otras palabras, cuanto más simple y fácil de usar sea una aplicación que construyamos con una interfaz de usuario (UX – UI), más complejo será el sistema backend detrás de ella.

La razón por la que tuve la idea de construir un sistema POS utilizando tecnología de reconocimiento de objetos con Deep Learning es: hace cuatro años llegué a España durante la pandemia de COVID sin conocer nada de español. Lamentablemente en muchos lugares no hablan inglés, entonces usaba el lenguaje corporal como señalar con el dedo para pedir comida: esto, aquello. Por otro lado, tengo muchos años de experiencia trabajando con sistemas POS (la más reciente, una empresa de software que co-fundé). Después de completar este Máster, especialmente con Business Intelligence, surgió una idea, ¿por qué no construir un sistema POS usando una nueva tecnología que sea el reconocimiento de objetos tanto para turistas como para locales? Los clientes se servirán ellos mismos usando dispositivos móviles para escanear imágenes de los platos que desean hacer el pedido. Además, todavía pueden realizar pedidos de forma tradicional según el menú del restaurante o realizar pedidos por voz.

El sistema POS tiene dos tipos: POS F&B (POS Alimentos y Bebidas) y POS Retails (Punto de Venta para comercios minoristas). En este trabajo, uno de los objetivos es poder construir una prueba de concepto o demostración de POS F&B (mock-up y flujo de trabajo), con el objetivo de reconocer platos preparados

a partir de las fotografías. También se incluirán recomendaciones de platos similares disponibles en la carta del restaurante. El sistema identificará automáticamente los platos, proporcionará información detallada sobre los ingredientes del plato y sugerirá platos similares en el menú. Además de integrar la funcionalidad de sugerir platos similares, mejorará la experiencia de uso, sugiriendo opciones alternativas en caso de que el plato solicitado no esté disponible.

La motivación para implementar este proyecto es ayudar a las empresas a mejorar la eficiencia empresarial. Específicamente, cuando los clientes no pueden encontrar el artículo exacto que buscan, las recomendaciones de artículos similares serán útiles, así como motivar a los clientes a reducir sus dudas a la hora de seguir realizando pedidos. Por otro lado, cuando el sistema sugiere platos similares o platos de acompañamiento, por ejemplo "al pedir este plato, los otros clientes suelen pedir un plato adicional abc, ...", esto ayuda a las empresas a obtener más ingresos.

Además, los clientes fieles o los clientes que se registraron para utilizar la Aplicación POS con historiales de pedidos, estarán más satisfechos si el sistema recomienda o almacena sus datos sobre los platos favoritos, en lugar de decirles a los camareros: “Como siempre, quiero este plato, …”. El cliente se sentirá recordado y atendido por el restaurante.

Y ciertamente, la aplicación gestionará los datos personales y serán seguros, respetando la privacidad del cliente.

El objetivo de este proyecto es solo una propuesta de desarrollo con una descripción detallada de los pasos a seguir para construir un sistema POS de alimentos y bebidas usando reconocimiento de objetos, en este caso reconocer platos (los pinchos) usando Deep Learning.

Objetivos del proyecto

Los principales objetivos de la tesis son:

2.1 Aplicación móvil/tablet – Versión del personal

La versión de la Aplicación de POS para gestionar las actividades comerciales de un restaurante o bar incluye funciones básicas como Pedido (pedir, abrir mesa, cerrar mesa, añadir platos, cambiar platos...), Inventario (actualizar en tiempo real), Pagos, Datos de acceso del cliente, etc., y la función Permiso permite el uso por cada rol de usuario: gerentes, contables, cajeros, chefs o camareros.

Además, cuando exista un plato sugerido por el sistema (recomendador) deseado por un cliente y que no esté disponible en la carta, en base a los ingredientes disponibles en cocina, el chef aceptaría preparar el nuevo plato según el pedido.

Y ciertamente, se construirá una aplicación web (backend) para gestionar todo el sistema, integrándose con las aplicaciones móviles vía API.

2.2 Aplicación móvil/tablet – Versión cliente

La aplicación versión Cliente será más sencilla que la versión del personal. Desde sus dispositivos móviles, los clientes pueden optar por iniciar sesión (cuenta registrada) o no iniciar sesión (incluye turistas, nuevos clientes o simplemente no quieren iniciar sesión).

La pantalla de pedido permitirá al cliente seleccionar el plato del menú (el plato se describirá detalladamente incluyendo los ingredientes), con la cantidad deseada y notas (si son alérgicos a algún ingrediente o quieren quitar algún ingrediente que no les gusta o agregar más cosas al plato, etc.,)

Hay varias formas de pedir comida desde la App:

- Escanear imágenes del menú o usar imágenes publicitarias del restaurante, el sistema identificará el plato si está disponible en el menú. El reconocimiento de imágenes utilizará una de las técnicas de Deep Learning, un algoritmo de aprendizaje supervisado como YOLO para reconocer el plato.

- Subiendo imágenes desde la aplicación, el sistema detectará el plato y permitirá al cliente pedirlo si está disponible.

- Seleccionando de forma tradicional desde el menú en pantalla.

- Pedir por voz (speech to text).

En caso de que el plato no esté disponible en el menú, el sistema sugerirá automáticamente alternativas similares, ayudando a los clientes a tener más opciones, estando más satisfechos mientras el restaurante obtendrá más facturación, un beneficio mutuo.

Para los clientes registrados, se almacenan todos los historiales de transacciones/pedidos. El cliente puede ahorrar tiempo a volver a pedir los platos que pide con frecuencia u obteniendo una recomendación de plato precisa. También es una forma de construir un sistema de gestión CRM, mejorar la eficiencia empresarial, satisfacer a los clientes, retener clientes leales e incluso tener más clientes futuros potenciales recomendados por los actuales.

Para los clientes no registrados o que no iniciaron sesión, las funciones de pedido no serán diferentes a los clientes que iniciaron sesión, pero debido a que no hay datos sobre su historial de pedidos, el sistema identificará artículos similares de manera genérica.

2.3 Tecnología

Sistema POS:

- Aplicación móvil/tablet: se pueden crear por separado los sistemas operativos iOS (Swift) y Android o utilizar marcos multiplataforma como React Native para ambos sistemas operativos.

- La aplicación web (backend) se puede crear mediante el lenguaje de programación PHP.

Nota: Para la demostración de POS de este proyecto, se creará con Flask, un marco micro web escrito en Python. Se clasifica como microframework porque no requiere herramientas ni bibliotecas particulares. No tiene una capa de abstracción de base de datos, validación de formularios ni ningún otro componente donde las bibliotecas de terceros preexistentes proporcionen funciones comunes (fuente Wikipedia [1]).

Sistema POS que utiliza Reconocimiento de Objetos con Deep Learning:

En este trabajo se utiliza YOLO como herramienta de reconocimiento de objetos. YOLO (You Only Look Once) es un algoritmo de reconocimiento de objetos en tiempo real con una precisión bastante alta, desarrollado por Joseph Redmon y Ali Farhadi en 2015.

Base de datos:

Las empresas pueden alquilar servidores de un proveedor de hosting/servidor, o utilizar servicios de almacenamiento en la nube de Google Cloud, AWS… o equipar sus propios servidores.

Nota: Para la base de datos de la demostración de POS se utilizará SQLAlchemy.

Seguridad de la información del cliente:

- Se utilizan métodos de autenticación y cifrado de datos para proteger la información de los clientes: protocolo SSL/TSL para almacenar y proteger los datos de comunicación entre la aplicación web y las aplicaciones móviles.

- Se utiliza un nombre anónimo como alias si los clientes no quieren proporcionar su nombre real.

- Cuando un cliente inicia sesión en el sistema, toda su información y su historial de transacciones se almacenarán en el sistema. Ya sea que se utilicen esos datos para entrenar al sistema para que identifique o sugiera con precisión los platos que los clientes desean, también debe contar con el consentimiento del cliente. Así, al registrar una cuenta en la aplicación, los clientes deben aceptar la política de privacidad, los términos de servicio, etc.

2.4 Recomendador

Para que una empresa funcione de forma eficaz, no sólo se basa en los ingresos, sino que también la satisfacción del cliente es la base para el desarrollo sostenible de cualquier negocio.

Además de los métodos tradicionales de atención al cliente, como llamar, enviar correos electrónicos, ofrecer promociones, descuentos y obsequios a clientes leales, desarrollar una aplicación con tecnología moderna y fácil de usar les brindará una gran experiencia. No sólo es nuevo y satisfactorio para los clientes, sino que también los hace sentirse importantes y agradecidos de que se les cuide "cada comida y cada sueño" como si fueran sus propias familias. Las herramientas de referencia (recomendador) que se integrarán en el sistema POS ayudarán a las empresas a lograrlo.

- Recomendar platos similares en caso de que no estén en la carta del restaurante o la cocina se quede sin ingredientes. Aplica tanto para clientes registrados como no registrados.

Conceptos teóricos

En el vibrante panorama empresarial actual, optimizar las operaciones y mejorar la experiencia del cliente es la clave del éxito. Para la industria de alimentos y bebidas, el sistema POS juega un papel importante y aporta muchos beneficios a las empresas. Este documento proporciona una solución, un flujo de trabajo que ayuda a construir un sistema POS combinado con tecnología de reconocimiento de objetos utilizando el algoritmo YOLO, que los sistemas POS convencionales aún no tienen o no han desarrollado.

3.1. Sistema POS F&B

El sistema POS es un sistema integral de gestión de ventas que combina hardware (máquina expendedora POS, terminal POS, impresora de facturas, etc.) y software para ayudar a las empresas a controlar las actividades comerciales de forma rápida y eficaz con funciones como pago, procesamiento de pedidos e impresión de facturas, ...

POS F&B (también conocido como software de gestión de restaurantes) es un software creado para controlar, soportar y optimizar la gestión de las operaciones en establecimientos que venden comida, café, ... de forma eficaz. El software proporciona muchas funciones adecuadas a las necesidades del negocio de restauración, como, por ejemplo:

- Realizar pedidos y facturar.
- Procesos de cocina.
- La gestión del inventario.
- Análisis de informes, ...

Además, se puede integrar con gestión de relaciones con el cliente (CRM), software de contabilidad, etc. Un sistema POS móvil es cuando un teléfono inteligente, tablet u otro dispositivo móvil actúa como terminal de ventas.

Descripción detallada del sistema POS

1. Gestión de usuarios con la función de crear, actualizar información y descentralizar operaciones sobre cada cuenta de usuario incluyendo gestión, contabilidad, cajero, camarero y chef.

Permiso según cada puesto:

Rol de gestión:

- Configuración del sistema, autorización de usuario (permiso).
- Gestión de empleados (agregar, editar, eliminar cuentas).
- Gestión de clientes (añadir, editar, eliminar cuentas).
- Administrar áreas de tabla (agregar, editar, eliminar áreas de tabla/tablas).
- Administrar artículos (agregar, editar, eliminar, actualizar precios, inventario).
- Ver sistema de informes (ver sección Informe).
- Funciones de pedido (ver sección Funciones de pedido).
- Administrar sucursales de restaurantes (si hay más de 1 sucursal).
- Gestión de inventarios (importación, exportación, inventario).
- Función de pantalla de cocina (cancelar plato, actualizar el estado del plato, como cocinado o terminado).

Rol de contable:

- Administrar artículos (agregar, editar, eliminar, actualizar precios, inventario).
- Gestión de inventarios (importación, exportación, inventario)
- Imprimir facturas.
- Pago.

Rol de camarero:

- Gestión de clientes (añadir, editar cuentas).
- Funciones de pedido (ver sección Funciones de pedido).
- Pantalla de cocina (solo se puede ver el estado de los platos en proceso, recién ordenados o completados).

Rol de chef o asistente:

- Función de pantalla de cocina (cancelar plato, actualizar el estado de los platos, como procesado o terminado).

2. Sistema de informes:

- Resumen de ventas.
- Pedido de venta.
- Informe de ventas por cliente.
- Informe de ventas por empleado.
- Informe de inventario.
- Informe de descuento.
- Informar de cancelación de pedido.
- Informe de facturación.
- Los pinchos más vendidos.
- Etc.,

3. Configurar y administrar el área de la mesa, incluidas funciones para crear una nueva mesa, actualizar ubicaciones y áreas de control.

4. Sistema de reserva de mesa (añadir, quitar mesa reservada)

5. Historial de pedidos (ver, reimprimir factura, repetir pedido)

6. El sistema administra y actualiza el menú, lo que permite a los administradores insertar datos manualmente:

- Agregar, editar o eliminar plato.
- Actualizar precio.
- Actualizar existencias.
- O crear subcategorías (si las tiene).

7. El usuario/cliente puede registrarse, iniciar sesión, cerrar sesión, editar información o eliminar su cuenta.

8. Las funciones de pedido permiten a los clientes/empleados operar en el sistema, incluyen:

- Interfaz principal.
- Seleccionar restaurante (si hay más de 1 sucursal).
- Seleccionar una mesa.
- Ordenar platos.
- Seleccionar la cantidad.
- Agregar notas.
- Buscar platos.
- Ver detalles del plato con ingredientes (advertencia de ingredientes que pueden causar alergias).
- Cancelar el plato.
- Descuentos.
- Confirmar pedido.
- Imprimir factura (Camarero/Cajero).
- Mover/Fusionar mesas (camarero).
- Pago (Camarero/Cajero).
- Historial de pedidos (ver, volver a pedir, solo el personal puede reimprimir la factura).

9. Sistema de pantalla para cocina que permite al personal saber qué nuevos pedidos se ordenaron, qué plato pertenece a qué mesa, y el personal de cocina puede actualizar el estado del plato, incluyendo: Cancelación, Procesamiento (cocinando) y Terminado.

10. El sistema permite al cliente realizar pedidos en la mesa e interactuar en la tablet o teléfono móvil. El sistema permite al cliente interactuar, llamando al personal y/o comprobando el estado del plato que ha elegido, si se está procesando o ya se ha completado.

3.2. YOLO – Real Time Object Detection

YOLO (You Only Look Once), un popular modelo de detección de objetos y segmentación de imágenes, fue desarrollado por Joseph Redmon y Ali Farhadi en la Universidad de Washington [4].

YOLOv11 es la última versión de YOLO de Ultralytics [31]. YOLOv11 logra una mayor precisión con menos parámetros gracias a los avances en el diseño de modelos y las técnicas de optimización. La arquitectura mejorada permite una extracción y un procesamiento de características eficientes, lo que da como resultado una mayor precisión media promedio (mAP) en conjuntos de datos como COCO y utiliza un 22% menos de parámetros que YOLOv8m. Esto hace que YOLO11 sea eficiente computacionalmente sin comprometer la precisión, lo que lo hace adecuado para su implementación en dispositivos con recursos limitados.

YOLOv11 presenta los componentes de bloque C3k2 (Cross Stage Partial con tamaño de kernel 2), SPPF (Spatial Pyramid Pooling - Fast) y C2PSA (Convolutional block with Parallel Spatial Attention). Estas nuevas técnicas mejoran la extracción de características y la precisión del modelo, lo que continúa con el linaje de YOLO de mejores modelos para casos de uso de detección de objetos en tiempo real. (Ver Detección de objetos mediante vídeo con YOLO11 aquí)

Aunque YOLO11 tiene una mejora en comparación con YOLOv8, también hereda características de YOLOv8. Debido a que, en esta investigación, todos los datos etiquetados son datos históricos, usar el entrenamiento del modelo de YOLOv8 es suficiente.

Como modelo de vanguardia (SOTA), YOLOv8 se basa en el éxito de versiones anteriores, introduciendo nuevas características y mejoras para mejorar el rendimiento, la flexibilidad y la eficiencia. El modelo YOLO que se puede utilizar para tareas de detección de objetos, clasificación de imágenes y segmentación de instancias. Las características principales de YOLOv8 incluyen aumento de datos en mosaico (Mosaic data), detección sin anclajes, un módulo C2f, un cabezal desacoplado y una función de pérdida modificada [15].

La siguiente imagen realizada por el usuario de GitHub RangeKing muestra una visualización detallada de la arquitectura de la red [18].

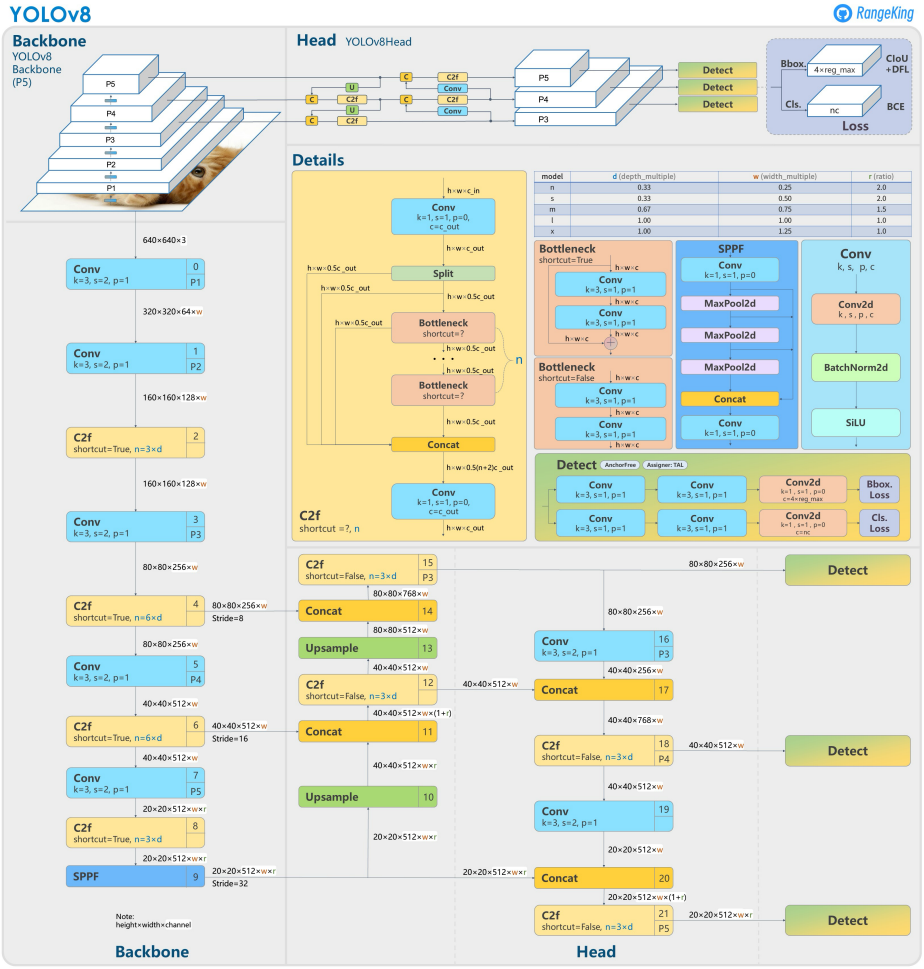


Figura 3.1: Arquitectura de YOLOv8 presentada por el usuario RangeKing.

Aumento de Datos de Mosaic

YOLOv8 utiliza un aumento de datos en mosaico que mezcla cuatro imágenes para proporcionar al modelo una mejor información de contexto. El cambio en YOLOv8 es que el aumento se detiene en las últimas diez épocas de entrenamiento para mejorar el rendimiento [15].

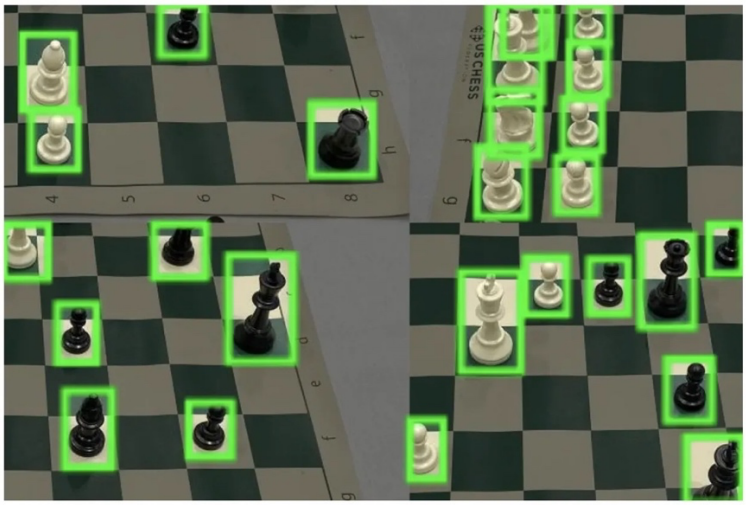


Figura 3.2: Aumento de Mosaic de fotografías de tableros de ajedrez – Roboflow.

Detección Anchor-Free

YOLOv8 cambió a detección sin anclajes (Anchor-free) para mejorar la generalización. El problema con la detección basada en anclajes es que los cuadros de anclaje predefinidos reducen la velocidad de aprendizaje de conjuntos de datos personalizados.

Con la detección sin anclajes, el modelo predice directamente el punto medio de un objeto y reduce la cantidad de predicciones del cuadro delimitador. Esto ayuda a acelerar la supresión no máxima (NMS), un paso de preprocesamiento que descarta predicciones incorrectas.

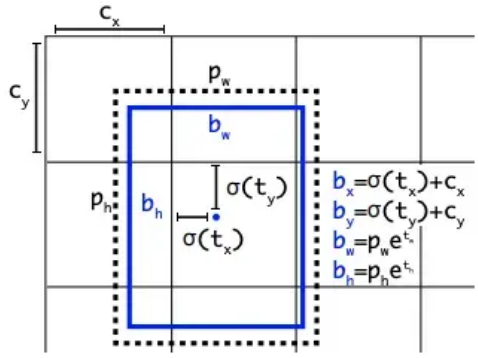


Figura 3.3: Visualización de una caja de anclaje en YOLO – Roboflow.

C2f Module

La columna vertebral del modelo ahora consta de un módulo C2f en lugar de uno C3. La diferencia entre los dos es que en C2f, el modelo concatena la salida de todos los módulos de cuello de botella. Por el contrario, en C3, el modelo utiliza la salida del último módulo de cuello de botella.

Un módulo de cuello de botella consta de bloques residuales de cuello de botella que reducen los costos computacionales en redes de aprendizaje profundo.

Esto acelera el proceso de entrenamiento y mejora el flujo de gradiente. [16].

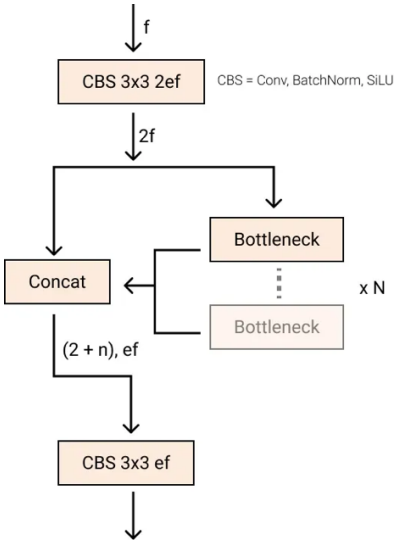


Figura 3.4: Nuevo módulo YOLOv8 C2f.

Decoupled Head

El diagrama anterior (Figura 3.4) ilustra que el cabezal ya no realiza clasificación y regresión juntas. En cambio, realiza las tareas por separado, lo que aumenta el rendimiento del modelo [17].

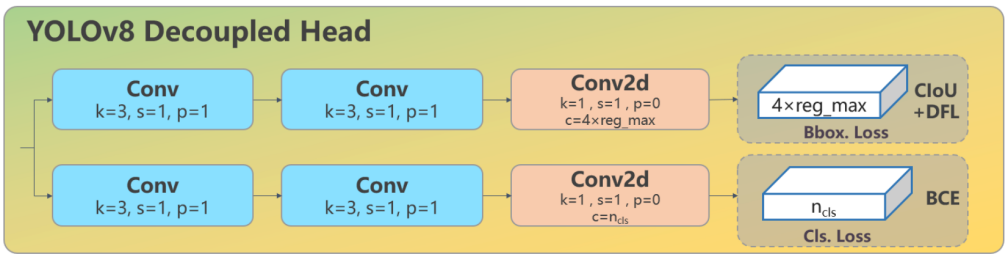


Figura 3.5: YOLOv8 Decoupled Head – MMLab.

Cálculo de pérdidas (Loss calculation)

El sesgo puede ocurrir porque la interfaz dividida separa las tareas de clasificación y regresión. Eso significa que el modelo puede localizar un objeto mientras clasifica otro.

La solución es incluir un punto de alineación de tareas, para que el modelo conozca las muestras positivas y negativas. La puntuación de alineación de la misión multiplica la puntuación de clasificación por la puntuación de Intersección en Unión (IoU - Intersection on Union). La puntuación de IoU corresponde a la precisión de la predicción del cuadro delimitador.

Según la puntuación de alineación, el modelo selecciona las k muestras positivas principales y calcula la pérdida de clasificación utilizando BCE (Binary Cross Entropy/Log Loss) y la pérdida de regresión utilizando IoU completo (CIoU) y pérdida de enfoque distribuido (DFL - distributed focus loss). La pérdida de BCE solo mide la diferencia entre la etiqueta real y la etiqueta prevista.

La pérdida de CIoU considera qué tan bien se relaciona el cuadro delimitador previsto con la etiqueta real sobre el punto central y la relación de aspecto. Por el contrario, la pérdida de enfoque distribuida optimiza la distribución de los límites del cuadro delimitador al centrarse más en muestras que el modelo clasifica erróneamente como falsos negativos [18].

Precisión de YOLOv8 COCO (Accuracy)

La investigación de YOLOv8 estuvo motivada principalmente por la evaluación empírica del punto de referencia COCO [5]. COCO (Common Objects in Context) es el punto de referencia estándar de la industria para evaluar modelos de detección de objetos [6].

Model	size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.2	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Tabla 3.1: Evaluación YOLOv8 COCO – Ultralytics.

La tabla 3.1 proporciona una comparación detallada de diferentes variantes del modelo YOLOv8, destacando su rendimiento y características computacionales.

- Modelo: Variantes del modelo YOLOv8:

  - YOLOv8n (Nano).
  - YOLOv8s (Pequeño).
  - YOLOv8m (Promedio).
  - YOLOv8l (Grande).
  - YOLOv8x (Extremadamente grande).

  La variación más pequeña y rápida es para entrenamiento en entornos con recursos computacionales limitados y obviamente tendrá menor precisión. Por el contrario, la gran variación tiene como objetivo entrenar modelos más grandes y proporcionar mayor precisión.

- Tamaño (size -pixels): es el tamaño de la imagen de entrada es de 640x640 pixels y se utiliza para entrenar y evaluar el modelo.

- Valor de mAP 50-95: la precisión promedio es una medida para evaluar la precisión de los modelos de detección de objetos, evaluados en diferentes umbrales de intersección sobre unión (IoU) del 50 % al 95 %. Los valores más altos indican un mejor rendimiento.

  Valores mAPval para la escala única de modelo único en el conjunto de datos COCO val2017. Reproducir usando "yolo val detect data=coco.yaml dispositivo=0"

- Velocidad: Velocidad de inferencia de modelos en diferentes plataformas de hardware:

  - CPU ONNX (ms): Velocidad de inferencia medida en milisegundos usando el formato ONNX en la CPU.
  - A100 TensorRT (ms): velocidad de inferencia medida en milisegundos usando TensorRT en la GPU NVIDIA A100.

  Acelere enormemente las imágenes de COCO val utilizando una instancia P4d de Amazon EC2. Reproducir por "yolo val detect data=coco8.yaml lote=1 dispositivo=0|cpu"

- Params (M): El número de parámetros del modelo, medido en millones. El modelo se entrena con muestras más complejas con parámetros más grandes pero requiere más recursos computacionales.

- FLOP (B): Los FLOP son operaciones de punto flotante medidas en miles de millones, lo que representa la complejidad computacional del modelo; el alto rendimiento viene con FLOP más altos.

3.3. Integración con el sistema POS

La familia de modelos YOLO pertenece a modelos de detección de objetos de una etapa que procesan la imagen completa en un solo paso de una red neuronal convolucional (CNN) [29].

Después de ser entrenado para reconocer objetos, YOLO procesará imágenes de platos (pinchos) cargadas o escaneadas desde el sistema POS para detectar y clasificar ese plato. El sistema recuperara de la base de datos relacional informaciones como nombre, precio, inventario, etc. Si ese plato ya no está en stock, se sugerirán platos similares. Después de que los clientes pidan comida, el sistema actualizará el inventario en tiempo real.

Recomendador

El recomendador utiliza la similitud del coseno al recomendar artículos similares basándose en listas de ingredientes de diferentes pinchos. Los pinchos que se consideran suficientemente similares se incluyen como recomendaciones cuando el stock de un pincho detectado es igual a 0.

La similitud del coseno mide la similitud entre dos vectores de un espacio producto interno. Se mide por el coseno del ángulo entre dos vectores y determina si dos vectores apuntan aproximadamente en la misma dirección. A menudo se utiliza para medir la similitud de documentos en el análisis de texto.

La similitud del coseno es una medida de similitud que se puede utilizar para comparar documentos o, por ejemplo, dar una clasificación de documentos con respecto a un vector determinado de palabras de consulta.

Técnicas y herramientas

Para desarrollar un sistema de Punto de Venta de Alimentos y Bebidas (POS F&B) utilizando reconocimiento de objetos con YOLOv8, es necesario realizar un proceso con muchos pasos de preparación diferentes, desde el diseño del flujo de trabajo, maquetas y procesamiento inicial de datos hasta los pasos finales de capacitación y pruebas.

4.1 Metodología

1. Deseño y prototipado (prototyping)

Describimos aquí los principales puntos a considerar:

- Diseñar el flujo de trabajo de un proceso para realizar pasos de pedido en la aplicación móvil para la versión del cliente y la versión del personal.

- Diseño detallado de maquetas de la aplicación POS Moblie (UX-UI) con una interfaz fácil de usar que incluye múltiples pantallas de pasos realizados en el sistema POS, como registrar un usuario, iniciar sesión, cerrar sesión, abrir una mesa, ver detalles de alimentos con advertencias rojas para algunos ingredientes que puedan causar alergias a los invitados, ordenar comida, cancelar comida, pago, reordenar comida, eliminar usuario.

- Crear un prototipo con UXPin para presentar visualmente el escenario de flujo de trabajo del sistema POS en dispositivos móviles. Para facilitar la visualización, las pantallas de diseño se diseñarán en tabletas en lugar de pantallas de dispositivos móviles. teléfono pequeño. De hecho, requiere 2 partes: construir el front-end con diferentes interfaces de diseño para tabletas, teléfonos o para la web, el back-end es construir una aplicación web, proporcionando API para aplicaciones móviles.

2. Procesamiento de datos (imágenes)

Ahora se muestran los siguientes puntos a considerar respecto al procesamiento de imágenes:

- Selección de 51 imágenes para 9 platos (los pinchos).
- Utilice Python para editar imágenes por lotes, incluyendo rotar, desenfocar, cambiar el tamaño, convertir a escala de grises y cambiar el tamaño a 640x640 píxeles para cumplir con los estándares YOLOv8.
- Cada imagen produce 8 imágenes nuevas diferentes:
    - blurred_resized_640_640
    - grayscaled_blurred_resized_640_640
    - grayscaled_resized_640_640
    - resized_640_640
    - rotated_180_grayscaled_blurred_resized_640_640
    - rotated_180_blurred_resized_640_640
    - rotated_180_resized_640_640
    - rotated_180_grayscaled_resized_640_640.

  En total se han creado 408 imágenes nuevas.
- Etiquetado de datos con Labelme: permite crear anotaciones para detectar, clasificar y segmentar conjuntos de datos de visión por computadora. Los pinchos se anotan con puntos y luego cada pincho se etiqueta con un número del 1 al 9. Después de etiquetar las imágenes, se guardarán como archivos JSON en la carpeta 'labelme_json_dir'.
- Convertir con labelme2YOLO: convierte el formato LabelMe JSON al formato de conjunto de datos de segmento YOLOv8.

3. Entrenamiento del modelo YOLOv8

Debido a que entrenar el modelo YOLOv8 requiere recursos como GPU y TPU potentes, usar Google Colab para entrenar durante 300 épocas será más rápido que una computadora personal y garantizará que el modelo esté completamente entrenado en el conjunto de datos.

Se necesitaron 1.665 horas para ejecutarse en Google Colab, mientras que las computadoras personales suelen tardar de 5 a 6 horas, dependiendo de la configuración de cada computadora.

En la Figura 4.1 se muestra una imagen del proceso de configuración de GoogleColab para este proyecto.

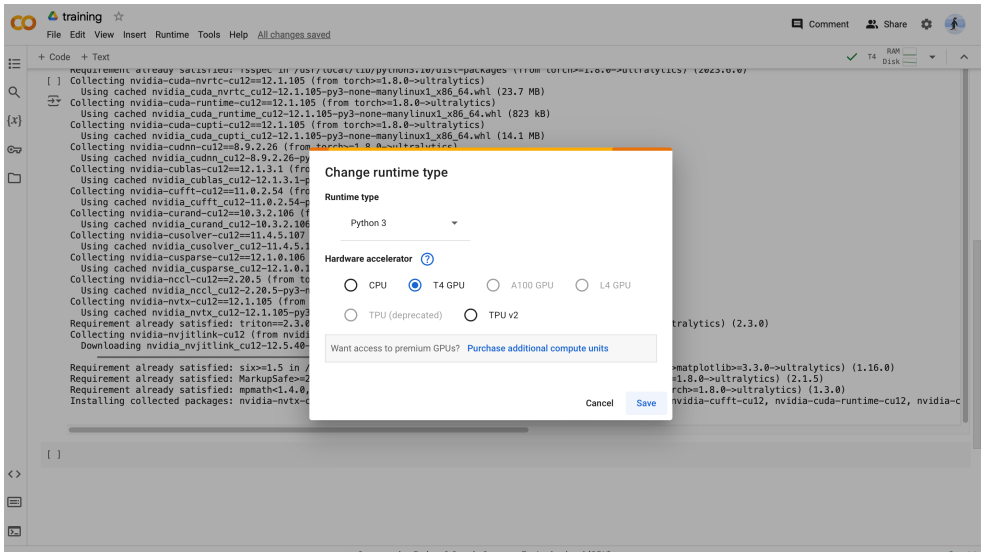


Figura 4.1: Configuración de Goole Colab.

- El entorno necesario para trabajar con bibliotecas de Deep Learning es el entorno Python 3.

- En la sección Acelerador de hardware, seleccione T4GPU. Acelerador T4 basado en la arquitectura Turing de NVIDIA, que incluye Tensor Cores para acelerar las operaciones de multiplicación de matrices, 2560 núcleos CUDA para manejar tareas computacionales en paralelo y 16 GB de memoria GDDR6, proporcionando un amplio espacio para manejar grandes conjuntos de datos y arquitecturas de modelos complejas.

4. Implementar modelos y desarrollar aplicaciones

Los puntos por considerar son:

- Crear una aplicación de demostración de POS utilizando Flask, un marco micro web escrito en Python.

- Integrar el modelo YOLOv8 entrenado (best.pt) en la demostración de POS para detectar y clasificar platos (los pinchos) cuando se carga, escanea u ordena una imagen por voz en tiempo real.
- Luego, el sistema POS recupera datos de esos pinchos (por número), la información relacionada, como nombre, precio, estado del inventario, se mostrará en la base de datos y permitirá a los clientes realizar pedidos mientras la comida aún está en stock.

5. Gestionar inventario y recomendar platos similares

El sistema verificará el estado del stock y, si el artículo está agotado, recomendará artículos similares en función de sus ingredientes u otros atributos utilizando similitud de coseno u otros algoritmos de recomendación.

4.2 Herramientas y Tecnologías

La Tabla 3.2 a continuación presentará las herramientas y la tecnología utilizadas para implementar este proyecto:

Herramientas	App	DB	Datos	Workflow Mockups	Prototipo	Memoria
HTML5	X					
CSS	X					
JavaScript	X					
Flask	X					
SQLAlchemy		X				
Python	X		X			
Similarity	X					
Labelme			X			
Labelme2YOLO			X			
YOLOv8	X		X			
YOLOv11	X		X			
Google Colab			X			
Google Drive			X			
Adobe Illustrator				X		
InVision					X	
UXPin					X	
Microsoft Word						X
Visual Studio code	X					

Tabla 3.2: Herramientas y tecnologías utilizadas en cada parte del proyecto.

1. Adobe Illustrator

Propósito de uso: Diseñar el flujo de trabajo para el proceso de pedido en la aplicación móvil para la versión de clientes (Figura 3.1) y la versión del personal (Figura 3.2) en el restaurante y la Experiencia de usuario e interfaz de usuario (UX-UI) para la versión móvil de POS F&B.

Illustrator crea formas básicas: mediante comandos y herramientas, copie y combine objetos para crear nuevas formas, seleccione y cambie partes dentro de objetos con la herramienta Selección y pinta los objetos. Proporciona muchos efectos 3D para dibujar formas, objetos y procesar imágenes relacionadas con el trabajo de diseño de páginas.

Hay muchos programas que se utilizan para diseñar UX-UI, como Sketch, InVision Studio, Figma, Balsamiq, ... Elijo utilizar Illustrator debido a muchas diferencias:

- Illustrator es un software que se centra en crear cosas nuevas. La IA se utiliza para crear productos basados en las ideas de los diseñadores.

- Illustrator es un software vectorial que permite realizar la mayoría de los requisitos básicos en el campo del diseño gráfico 2D. El funcionamiento es muy fácil y rápido, el dibujo es muy fluido y la edición de objetos es rápida. Buen soporte para diseño de página, combinación de colores, ajuste de tamaño y muchos otros efectos complejos...

- Exporte a muchos formatos de archivo compatibles, como .AI y .EPS.

- Illustrator destaca por su capacidad para copiar objetos. Adobe Illustrator es un software de diseño y edición de gráficos vectoriales desarrollado y comercializado por Adobe.

Licencia: Sí.

2. Invision & UXPin

Propósito de uso: Creación de prototipos y flujos de trabajo interactivos.

InVision es una plataforma popular de diseño y creación de prototipos gracias a muchas ventajas excepcionales. Proporciona un conjunto completo de herramientas que simplifican y optimizan el proceso de diseño. InVision permite

a los diseñadores compartir, comentar, probar e implementar sus diseños de forma rápida y eficiente.

Se centra en la experiencia del usuario, ayudando a simular y probar diseños antes de su implementación, mejorando la calidad y usabilidad del producto. Con una interfaz amigable y fácil de usar y altas capacidades de personalización, InVision permite a los diseñadores familiarizarse rápidamente y ser flexibles en su flujo de trabajo.

Al integrarse bien con herramientas de diseño como Sketch, Adobe XD y Figma, InVision admite eficazmente flujos de trabajo ágiles, lo que aumenta la flexibilidad y la respuesta rápida.

Funciones como comentarios, opiniones y tareas facilitan la colaboración y la comunicación dentro del equipo de diseño.

InVision garantiza la seguridad y el diseño de los datos, evita la fuga de información y protege la privacidad.

InVision, multiplataforma, permite el acceso y el trabajo a través de computadoras, teléfonos móviles y tabletas, brindando flexibilidad y conveniencia para el trabajo remoto o sobre la marcha.

El 30 de diciembre de 2024, InVision fue vendido a Miro, una plataforma de colaboración digital diseñada para facilitar la comunicación y gestión de proyectos de equipos remotos y distribuidos. Todos los flujos de trabajo y UI/UX son compatibles con InVision para exportación y transferencia a Miro, Sketch o UXPin. Aunque Miro cuenta con Prototype, su interfaz no es intuitiva y es de pago. UXPin, una plataforma similar para la creación de flujos de trabajo y prototipos, es una alternativa perfecta.

Licencia: Sí (InVision) & No (UXPin).

3. Python

Propósito de uso:

- Procesamiento de datos: edición por lotes de imágenes como rotación, desenfoque, cambio de tamaño y escala de grises. y cambie su tamaño a 640x640 píxeles.

- Combina con las herramientas labelme y labelme2YOLO para convertir el formato JSON de LabelMe al formato de conjunto de datos YOLOv8.

- A través de Google Colab, utiliza el algoritmo YOLO para entrenar imágenes etiquetadas.

- Crear una aplicación POS de demostración con el marco Flask.

Python es un lenguaje de programación multiplataforma, de código abierto y de alto nivel. Python fue introducido por Guido van Rossum en 1991 y ha pasado por 3 etapas de desarrollo diferentes correspondientes a las versiones, siendo la última actualmente la versión 3x de Python (3.12.3 el 9 de abril de 2024). Python tiene una sintaxis clara y concisa, lo que facilita el aprendizaje y el uso del lenguaje (fuente Wikipedia).

Licencia: Open-source.

4. Labelme

Propósito de uso: Etiquetar imágenes con anotaciones de puntos y conviértalas a formato JSON.

LabelMe es un proyecto creado por el Laboratorio de Ciencias de la Computación e Inteligencia Artificial del MIT que proporciona un conjunto de datos de imágenes digitales con anotaciones. El conjunto de datos es dinámico, de uso gratuito y abierto a la contribución pública [12].

LabelMe permite crear anotaciones para conjuntos de datos de visión por computadora de detección, clasificación y segmentación de objetos. Podemos dibujar anotaciones usando polígonos, rectángulos, círculos, líneas y puntos.

LabelMe proporciona una interfaz gráfica a través de la cual podemos anotar imágenes para:

- Detección de objetos, mediante cuadros delimitadores y polígonos.
- Segmentación de instancias.
- Segmentación semántica.
- Clasificación.Label images with point annotations, convert to JSON format.

Licencia: Open-source.

5. Labelme2YOLO

Propósito de uso: Convertir el formato LabelMe JSON al formato de conjunto de datos YOLOv8 para el entrenamiento previo del modelo YOLO.

Labelme2YOLO es una poderosa herramienta para convertir el formato JSON de LabelMe al formato de conjunto de datos YOLOv5. Esta herramienta también se puede utilizar para conjuntos de datos de segmentación YOLOv5/YOLOv8. Si ya hemos realizado el conjunto de datos de segmentación con LabelMe, es fácil usar esta herramienta para ayudar a convertir el conjunto de datos en formato YOLO. [13].

Licencia: No.

6. YOLOv8

Propósito de uso:

- Detección de objetos: Después del entrenamiento con 408 imágenes editadas, se utilizó el modelo (best.pt) para reconocer platos (los pinchos) con alta precisión.

- Integrar el modelo de capacitación en el sistema POS: El modelo YOLOv8 entrenado (con 300 épocas) se integrará en el sistema POS, lo que permitirá a los usuarios usar dispositivos móviles para cargar o escanear imágenes de platos en el sistema para la identificación de 'los pinchos' y reconocimiento, ayúdelos a realizar pedidos de forma rápida y sencilla.

YOLO, o "Sólo miras una vez", es un algoritmo de detección de objetos y un modelo de segmentación de imágenes en visión por computadora, desarrollado por Joseph Redmon y Ali Farhadi en la Universidad de Diseño y Creación de Prototipos [16].

La característica especial de YOLO radica en la forma en que realiza predicciones sobre cuadros delimitadores y probabilidades de objetos en una sola pasada de imagen.

En el pasado, los algoritmos de detección de objetos solían utilizar clasificadores previamente entrenados para identificar objetos después de generar posibles regiones de interés. Sin embargo, YOLO lo hace de manera diferente al hacer predicciones durante todo el proceso en una sola ejecución, desde localizar objetos hasta clasificarlos.

Con este enfoque innovador, YOLO ha logrado mejoras significativas y superó a otros algoritmos de detección de objetos, especialmente al garantizar el rendimiento en tiempo real.

YOLOv8 es la última versión de la línea de modelos YOLO. Los modelos YOLO están previamente entrenados en grandes conjuntos de datos como COCO e ImageNet. Esto les permite proporcionar predicciones extremadamente precisas con las clases en las que han sido entrenados y aprender nuevas clases con relativa facilidad.

Los modelos YOLO también tienen una velocidad de entrenamiento más rápida, con alta precisión y un tamaño de modelo pequeño. Se pueden entrenar en GPU únicas, lo que los hace más accesibles para los desarrolladores.

Anunciado a principios de 2023, YOLOv8 aportó muchas ventajas en comparación con su predecesor, como la detección sin anclajes, la introducción de la capa de convolución C3 y la mejora del mosaico [17].

Licencia: No.

7. Google Colab

Propósito de uso: Entrenar el modelo YOLO con conjuntos de datos procesados rápidamente sin tener que invertir en una computadora de alta configuración.

Colaboratory, también conocido como Google Colab, es un producto de Google Research que permite ejecutar comandos Python en la plataforma en la nube.

Colab no requiere instalación ni configuración potente del ordenador, todo puede ejecutarse a través del navegador, permitiéndonos utilizar recursos del ordenador desde CPU, GPU o TPU de alta velocidad (gratis si la opción predeterminada es CPU).

La interfaz de Google Colab es muy similar a Jupyter Notebook, una herramienta que ayuda a ejecutar cada línea de comando de Python visualmente y verificar los resultados del comando en el acto.

Bibliotecas de Python populares como: Pandas, Keras, Tensorflow, Matplotlib, Numpy están todas preinstaladas. Todas estas son bibliotecas adecuadas para el preprocesamiento de datos, el análisis de datos y el aprendizaje automático.

Para entrenar el modelo YOLOv8 con un conjunto de datos procesados de 408 imágenes en 300 épocas, podemos usar Jupyper Notebook o Anaconda en una computadora personal, o Visual Studio Code. Pero usar Google Colab con soporte para GPU es más rápido que una PC normal y garantiza que el modelo esté completamente entrenado en el conjunto de datos. Solo tardó 1.665 horas en ejecutarse en Google Colab.

Licencia: Sí.

8. Flask

Propósito de uso:

- Construir la aplicación de demostración POS (funciones de pedido), integrando detección de artículos (detección de objetos) y recomendación de artículos similares.

- Interacciones del usuario: registro, inicio de sesión, cierre de sesión, eliminar cuenta.

Flask es un marco micro web escrito en Python. Se clasifica como microframework porque no requiere herramientas o bibliotecas particulares. No tiene una capa de abstracción de base de datos, validación de formularios ni ningún otro componente donde las bibliotecas de terceros preexistentes proporcionen funciones comunes (fuente Wikipedia [1]).

Fue desarrollado por Armin Ronacher, líder del Grupo Internacional de Entusiastas de Python (POCCO). Básicamente se basa en el kit de herramientas WSGI y el motor de plantillas Jinja2. WSGI es un acrónimo de interfaz de puerta de enlace del servidor web, que es un estándar para el desarrollo de aplicaciones flask del marco web Python. Se considera la especificación para la interfaz universal entre el servidor web y la aplicación web. Jinja2 es un motor de plantillas web que combina una plantilla con una determinada fuente de datos para representar páginas web dinámicas. En Easy Language, combina una plantilla (el diseño de la página) con datos (la información específica que desea mostrar) para crear una página web dinámica [26].

Licencia: Open-source.

Aspectos relevantes del desarrollo del proyecto

Esta sección describirá cómo se utilizan las tecnologías y herramientas paso a paso para desarrollar el proyecto "Un sistema de punto de venta de alimentos y bebidas (POS F&B) utilizando el reconocimiento de objetos con YOLOv8".

5.1 Deseño y Prototipado

La primera fase es la creación de conceptos, el diseño del flujo de trabajo y el diseño del sistema POS utilizando Adobe Illustrator [28] para crear diseños de experiencia de usuario (UX) e interfaz de usuario (UI). La función de diseño basado en vectores de Illustrator ayuda a diseñar elementos de la interfaz de usuario con precisión, lo que garantiza una interfaz limpia, fácil de leer y fácil de usar.

1. Flujos de trabajo (workflows) y diseño UX-UI

El proceso de pedido en la aplicación móvil tiene 2 versiones:

- Versión cliente: Permite que los clientes se registren, inicien sesión en su cuenta o inicien sesión como invitado para ordenar el plato.

- Versión para personal: a la versión para personal se le otorgará permiso para cada rol, como cajero, camarero, chef o gerente.

Las siguientes figuras son el proceso de pedido en la aplicación móvil para la versión del cliente (Figura 5.1) y la versión del personal (Figura 5.2) en el restaurante.

Proceso de pedido – POS F&B en móvil - Versión cliente

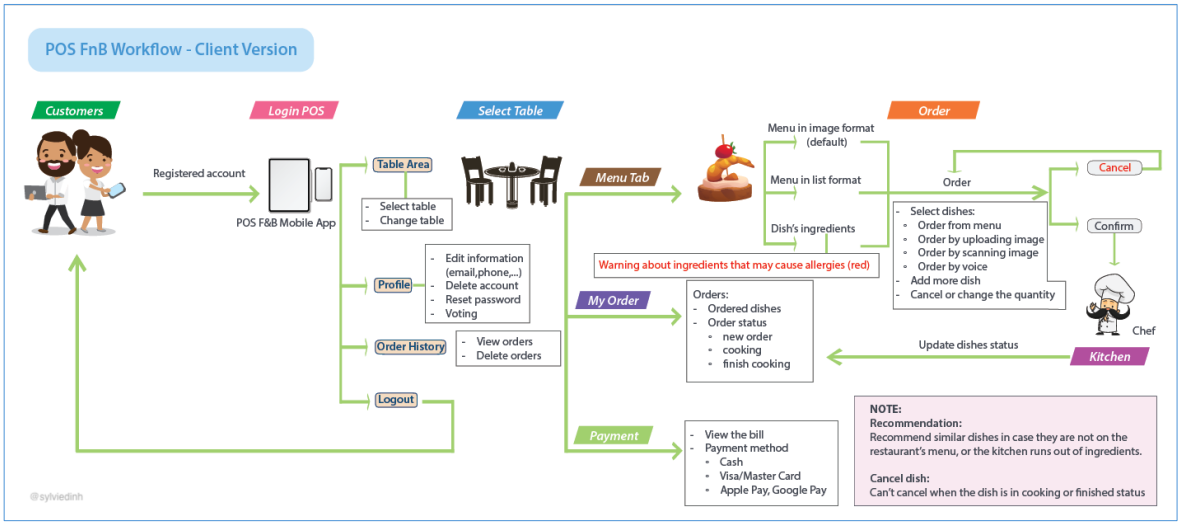


Figura 5.1: Proceso de pedido en la versión de la aplicación móvil para clientes.

Proceso de pedido – POS F&B en móvil - Versión de personal

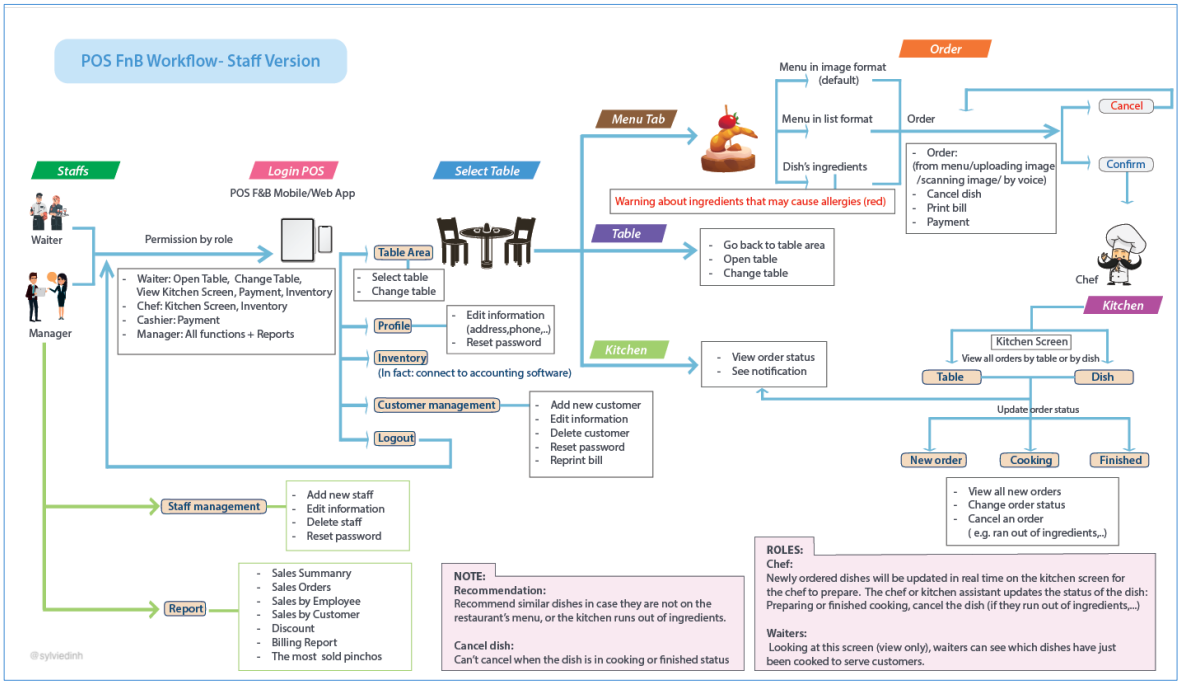


Figura 5.2: Proceso de pedido en la versión de la aplicación para el personal.

Versión Cliente

Los clientes pueden realizar sus pedidos mediante el sistema POS móvil, versión para cliente. En esta aplicación, los clientes pueden registrar una cuenta (Figura 5.3), iniciar sesión, actualizar información, cerrar sesión y eliminar su cuenta (Figura 5.4).

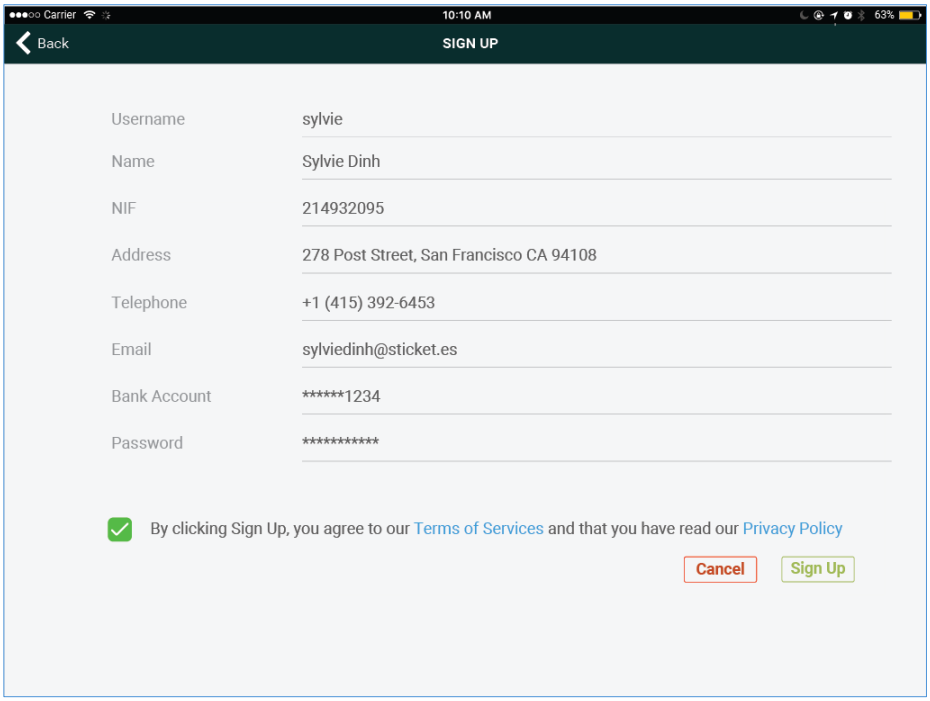


Figura 5.3: Pantalla de registro de cliente.

Registrarse e iniciar sesión en el sistema significa que el cliente acepta los términos y la política de privacidad del restaurante. La ventaja es que los pedidos se guardarán en el historial de pedidos, los clientes podrán recuperarlos y volver a realizarlos.

Permitir a los clientes eliminar sus cuentas directamente en la aplicación es conveniente y respeta la privacidad de los clientes cuando ya no quieren utilizar el servicio.

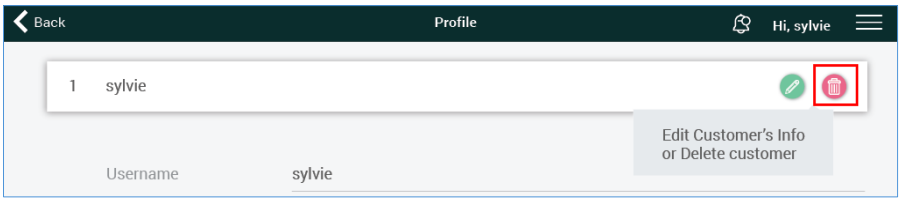


Figura 5.4: Eliminar una cuenta.

Para realizar pedidos, los clientes pueden iniciar sesión en su cuenta o iniciar sesión como invitado, o pueden realizar pedidos a un camarero. Luego de abrir la aplicación, el cliente seleccionará y abrirá una mesa vacía (color verde); el color rojo son las

mesas ocupadas (ya sean ocupadas o reservadas), el color azul son las que acaban de seleccionar.

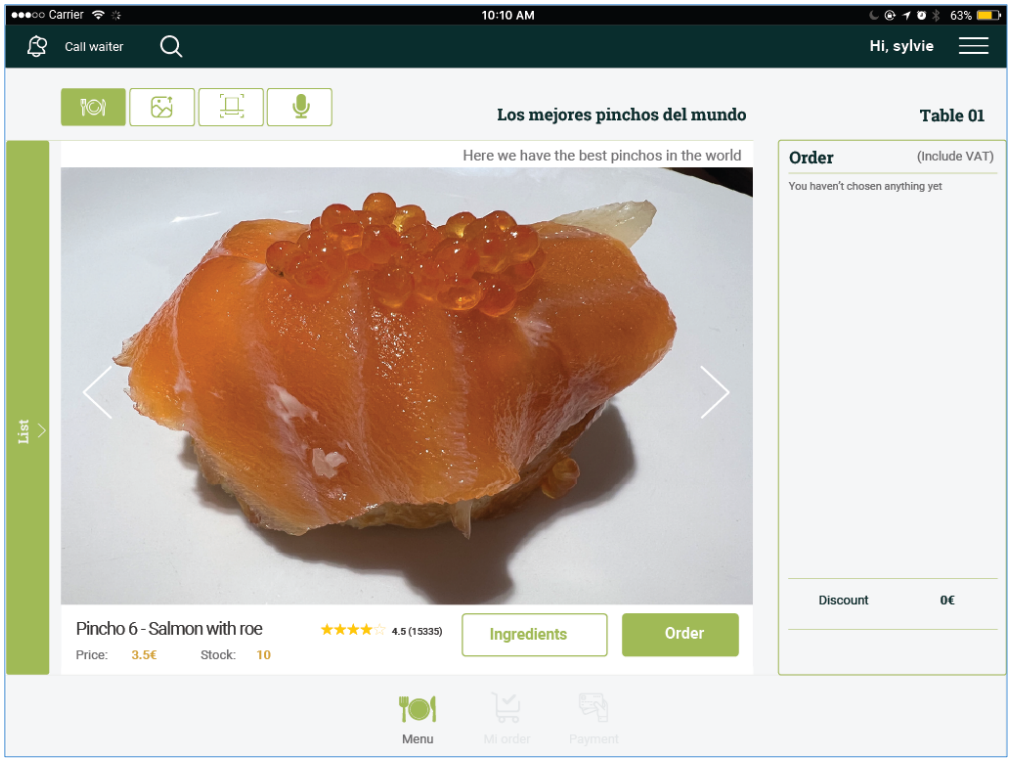


Figura 5.5: Pestaña de menú después de seleccionar una mesa.

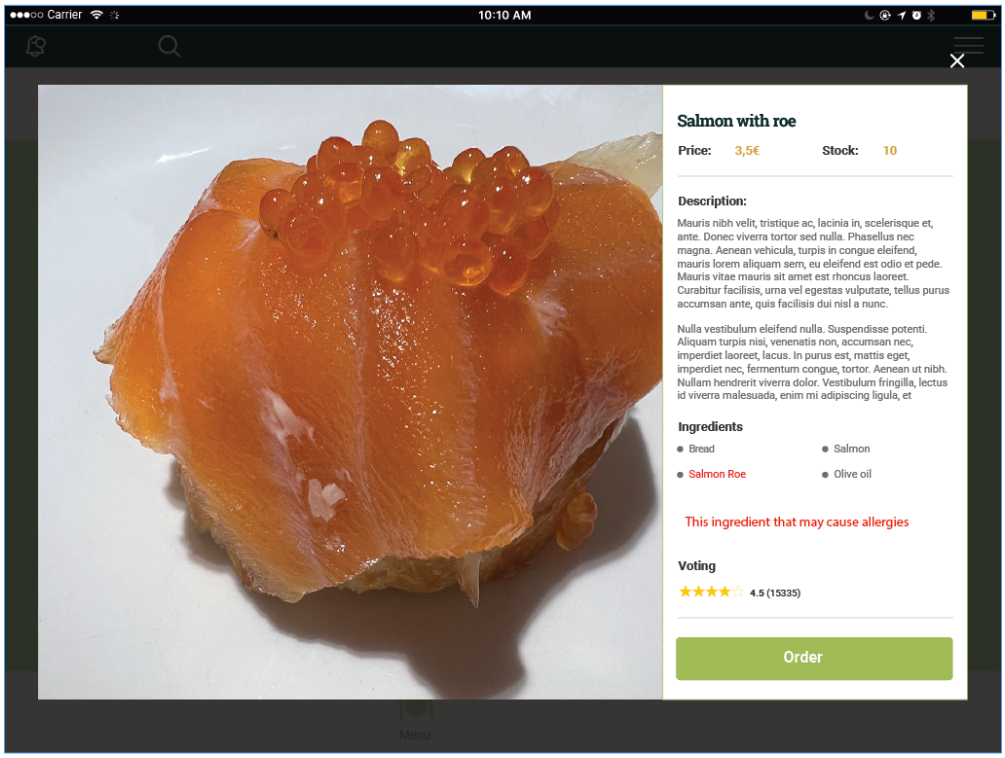


Figura 5.6: Ingredientes del plato con advertencia alérgica.

Después de abrir la mesa, la pantalla predeterminada es la pestaña "Menú" (Figura 5.5) que permite realizar pedidos desde el menú, con platos que muestran imágenes, stock, cantidad y se mostrarán los ingredientes del plato que pueden causar alergias (Figura 5.6). La marca roja tiene un tamaño de fuente mayor que la descripción para advertir a los clientes.

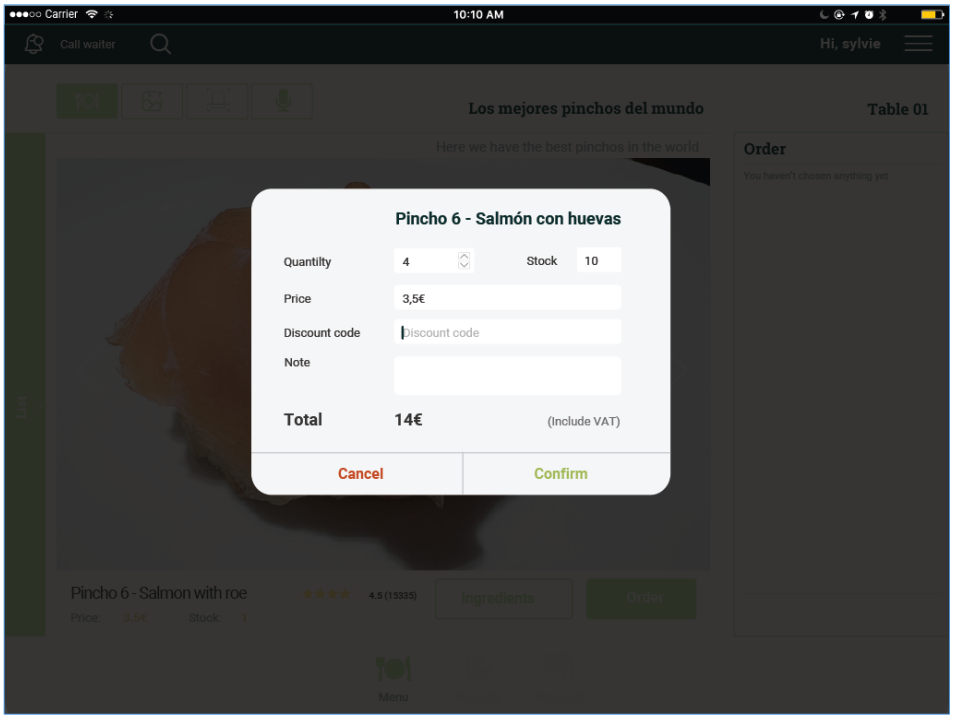


Figura 5.7: Confirmar un pedido.

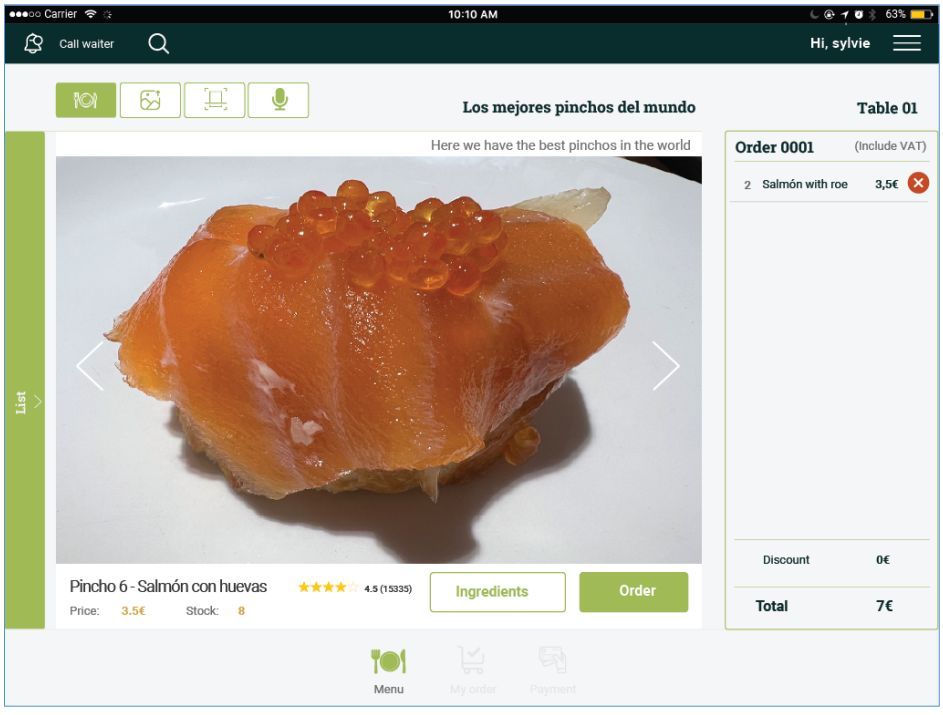


Figura 5.8: Lista de pedidos con factura temporal.

Los clientes piden comida con el número deseado (Figura 5.7), la factura temporal (Figura 5.8) se mostrará en el lado derecho de la pantalla web o del tablet (para teléfonos con pantalla pequeña, desplácese hacia abajo para ver la factura).

Los clientes pueden cancelar un pedido o cambiar la cantidad cuando se acaba de realizar el pedido.

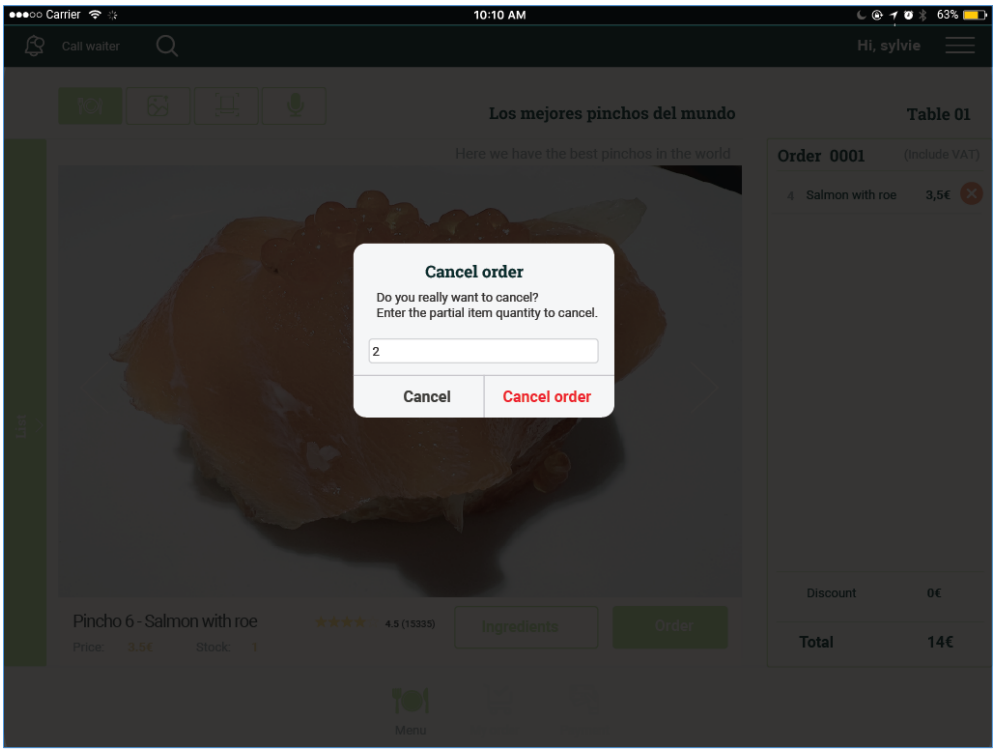


Figura 5.9: Cancelar un pedido.

Además, los clientes pueden realizar pedidos por voz.

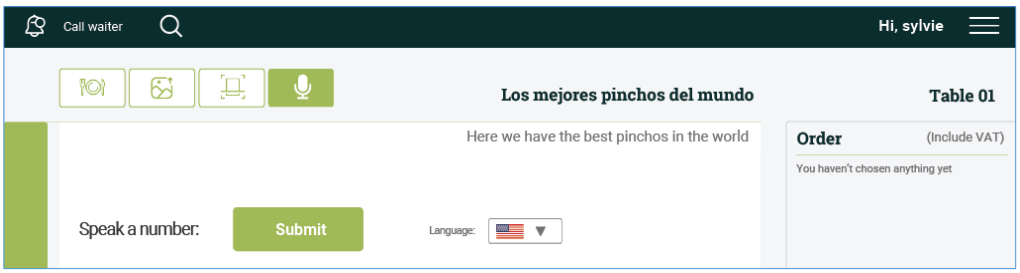


Figura 5.10: Pedir por voz.

La diferencia de este proyecto es que crea ideas para que los clientes realicen pedidos cargando (Figura 5.13) o escaneando imágenes (Figura 5.11). El sistema identificará el plato y mostrará la información del mismo casi en tiempo real para que los clientes puedan realizar el pedido mientras el plato aún esté disponible. Para hacer esto, la

Sección 5.3 - Entrenamiento del modelo describirá con más detalle el entrenamiento del modelo YOLO en el reconocimiento de objetos.

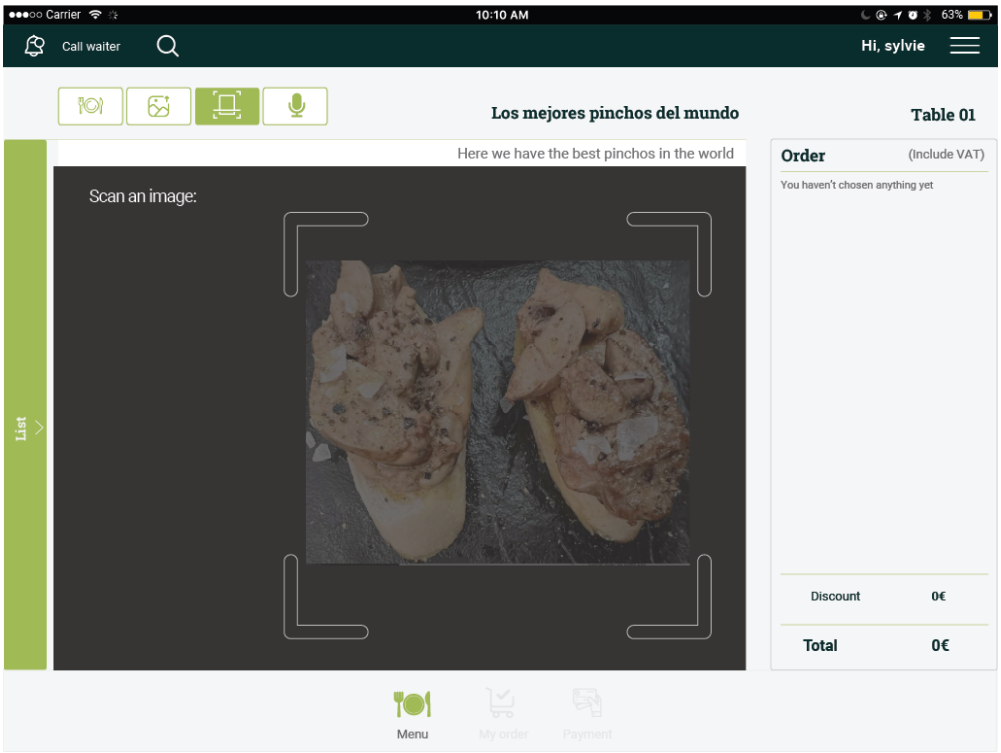


Figura 5.11: Realiza el pedido escaneando una imagen.

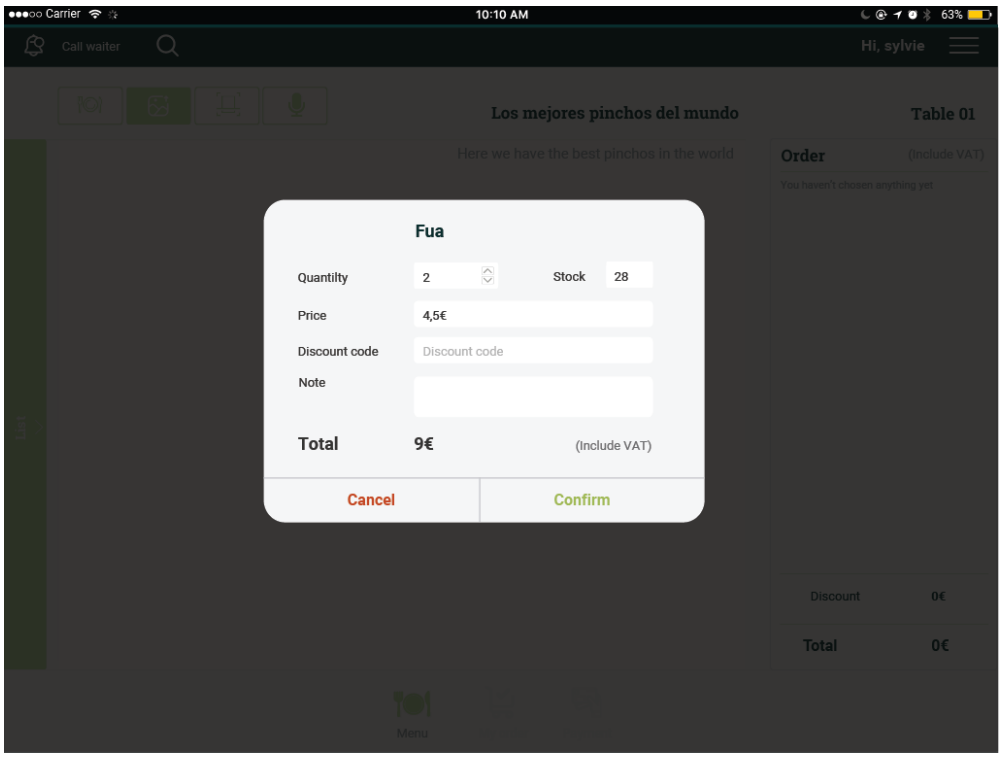


Figura 5.12: Identificación de pinchos en tiempo real.

Además, el sistema te sugerirá platos similares si ese plato está agotado.

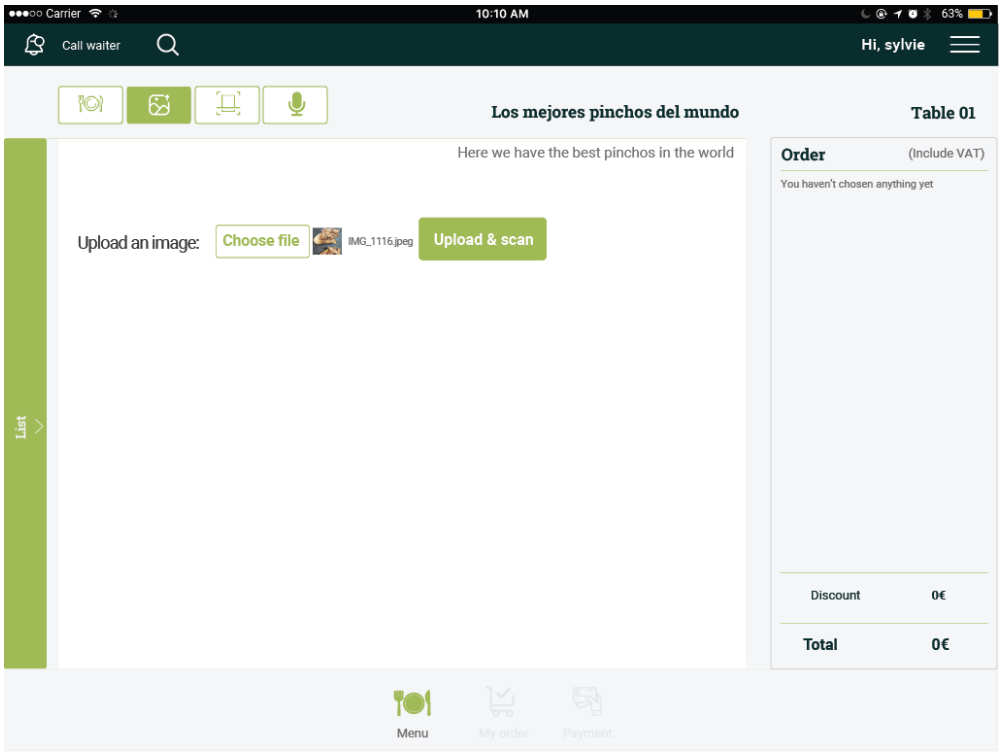


Figura 5.13: Realiza el pedido subiendo una imagen.

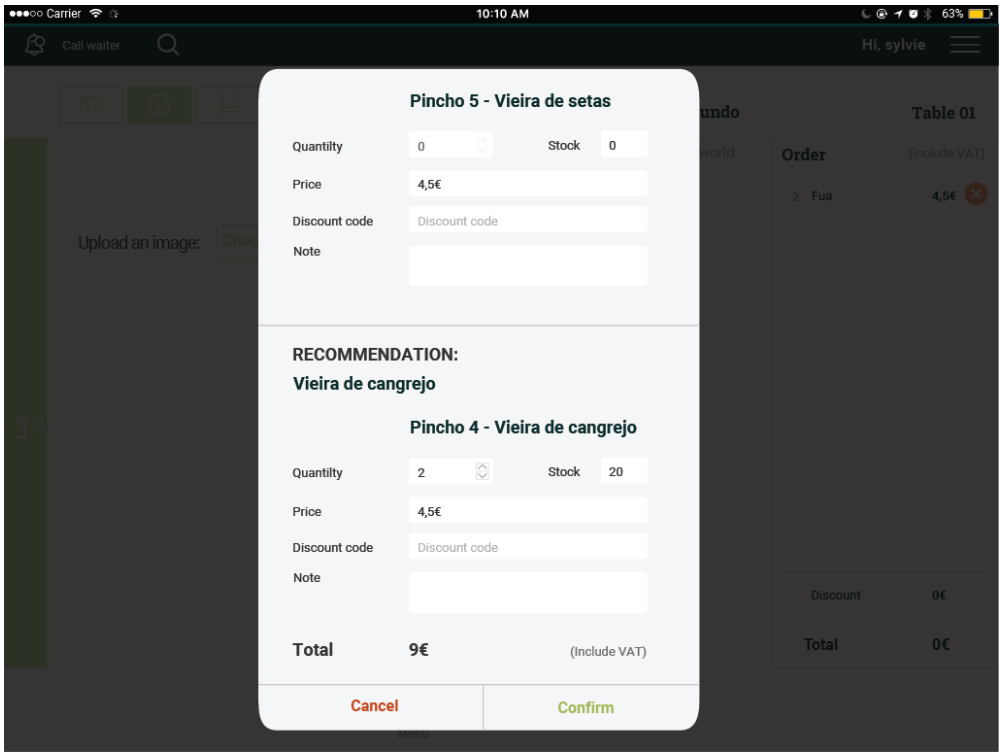


Figura 5.14: Recomendación de pinchos similares.

Otra opción es tener un botón "Campana" para llamar al camarero (opcional) cuando los clientes necesiten algo.

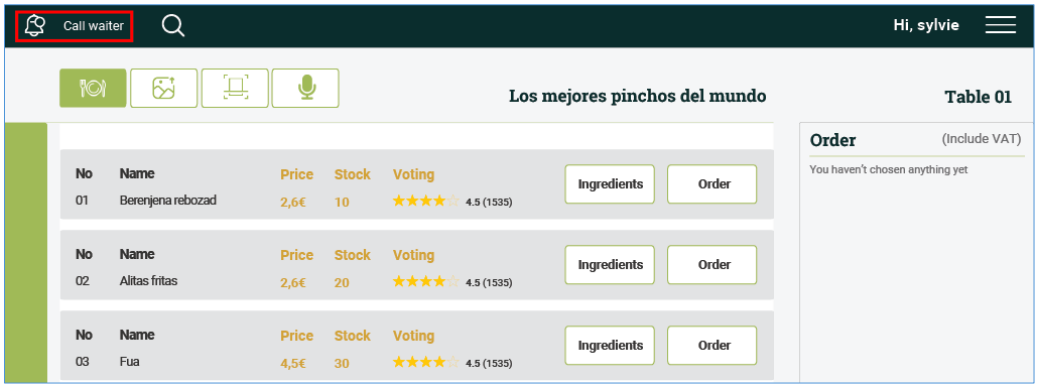


Figura 5.15: Botón de llamada al camarero.

La segunda pestaña "Mi pedido" de la pantalla muestra el estado de los platos, como platos recién pedidos, platos en proceso y platos completados. Para platos en proceso (cocción) y platos terminados, los clientes no pueden cancelar pedidos. Quizás en realidad el restaurante pueda ser más flexible y permitir la cancelación de pedidos (muchos casos requieren permiso de la dirección).

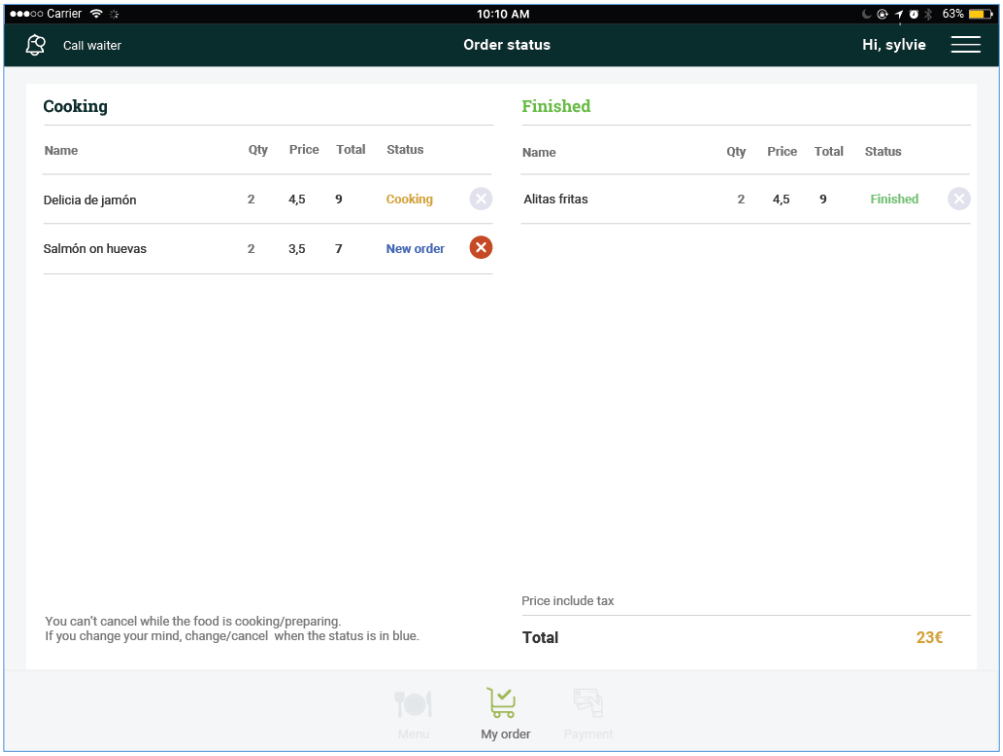


Figura 5.16: Estado de pedido.

La tercera pestaña de la pantalla “Pago” permite a los clientes revisar facturas y pagos, seleccionar métodos de pago y el camarero solo necesita acercar el Terminal POS a la mesa del cliente.

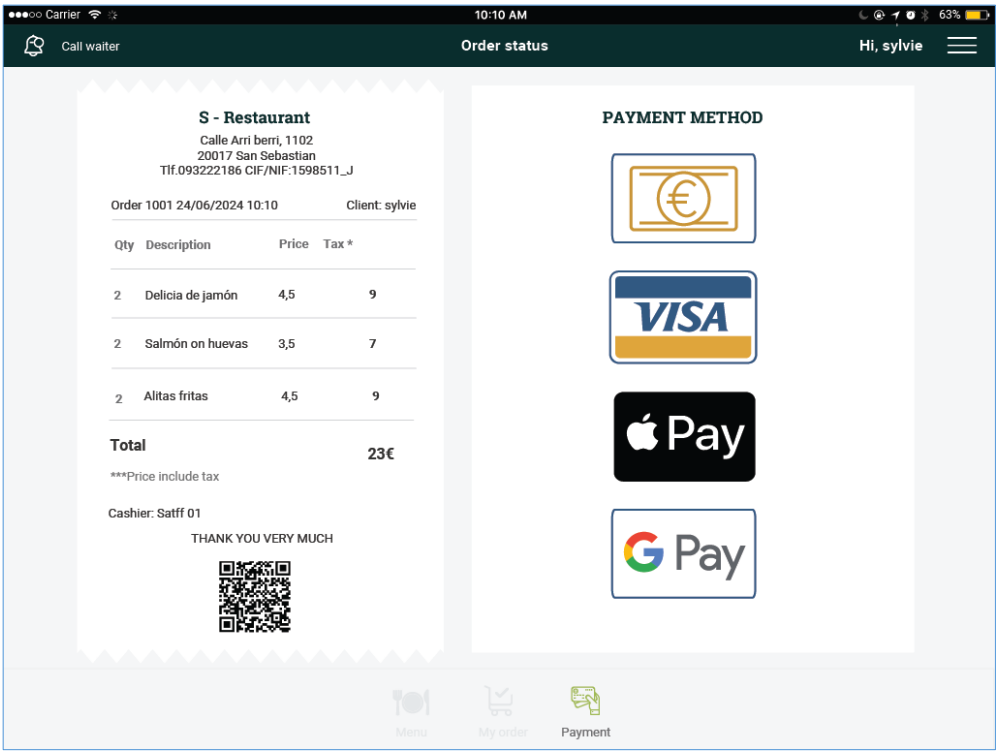


Figura 5.17: Revisión de factura y elección de método de pago.

En la parte superior derecha hay un botón de menú oculto, que incluye:

- “Perfil” donde los clientes pueden actualizar información y eliminar cuentas (Figura 18).
- “Historial de pedidos” para revisar el historial de pedidos y poder reordenar una factura (Figura 5.19) con artículos familiares.
- Y “Cerrar sesión” del sistema.

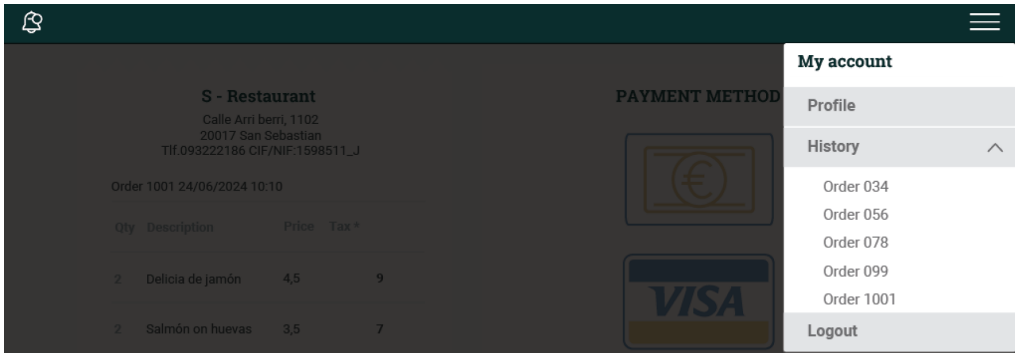


Figura 5.18: Menú de configuración del cliente.

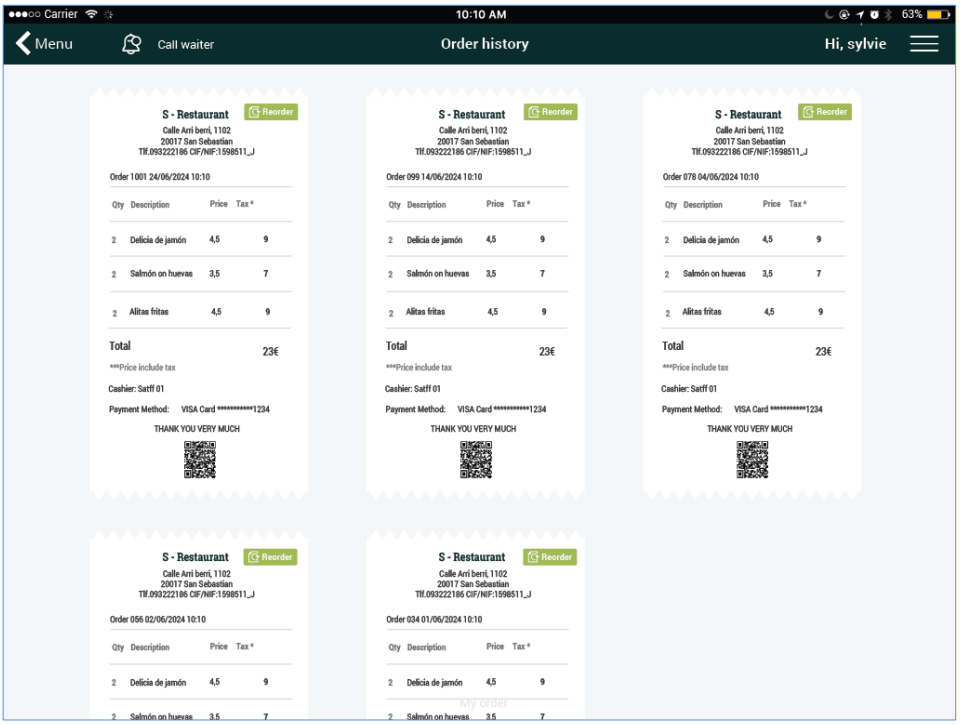


Figura 5.19: Historial de pedidos del cliente.

Versión personal

Funciones de pedido:

Similar al POS F&B - versión cliente, el personal puede ordenar desde el menú, por voz, cargando o escaneando imágenes, se recomendará un plato similar cuando el plato deseado ya no esté en stock, pero existen algunas diferencias:

- Al seleccionar y abrir una mesa vacía (verde) para un cliente, el personal puede cambiar a otra mesa; El color rojo indica mesas ocupadas o el color amarillo es mesa reservada. En caso de que un cliente llame para cancelar una mesa reservada, el personal cancelará esa mesa.

- Hay 3 pestañas en la pantalla principal después de iniciar sesión: Menú, Mesa y Cocina.

    - La pestaña “Menú” predeterminada (Figura 5.21) se muestra como una lista de platos para que el personal pueda ordenar rápidamente. Existe la opción de ver el menú por imagen en caso de que el cliente quiera ver imágenes del plato y los ingredientes que lo acompañan.

    - La pestaña "Mesa" es la gestión de mesas (Figura 20) que permite al camarero abrir esta pestaña para pasar a servir otras mesas.

- La pestaña "Cocina" (Figura 5.23) tiene el propósito de ver el estado de los platos: recién ordenados, cocinándose o completados para ayudar a los camareros a saber qué platos se acaban de preparar en qué mesa servir a los clients.

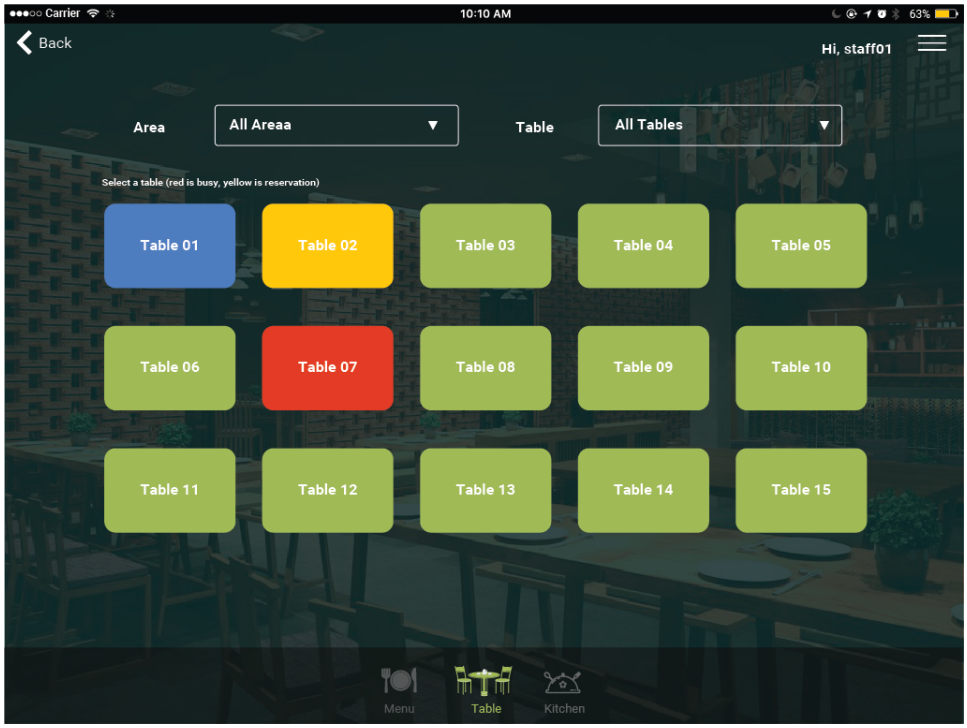


Figura 5.20: Gestión de mesa.

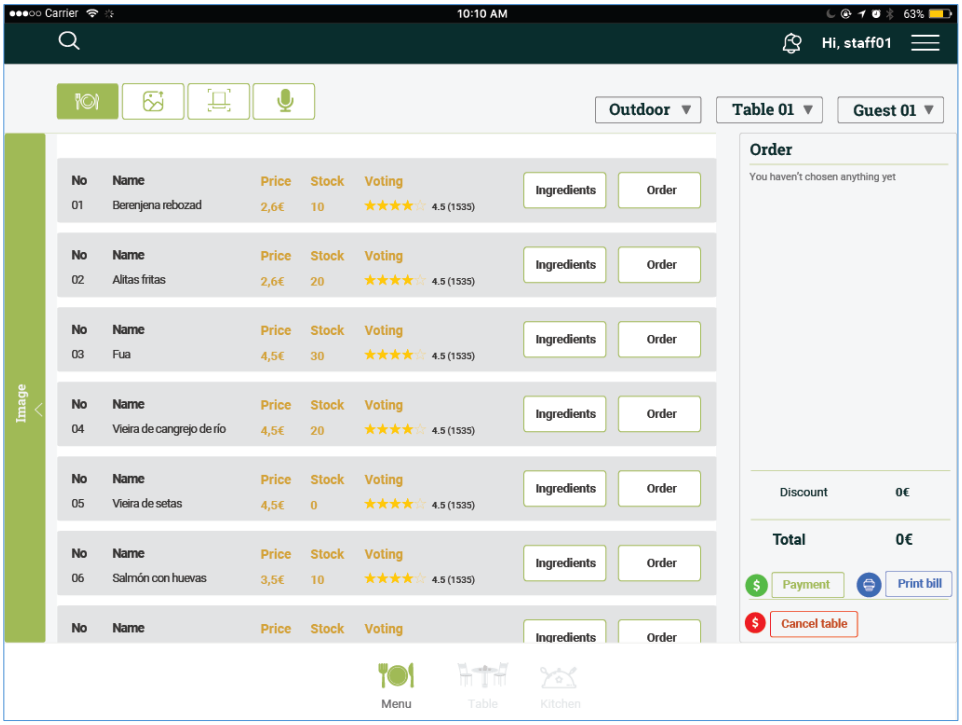


Figura 5.21: Pestaña Menú en el sistema POS – versión para personal.

- Además, existe una sección para seleccionar el nombre del cliente o invitado luego de abrir la nueva mesa.

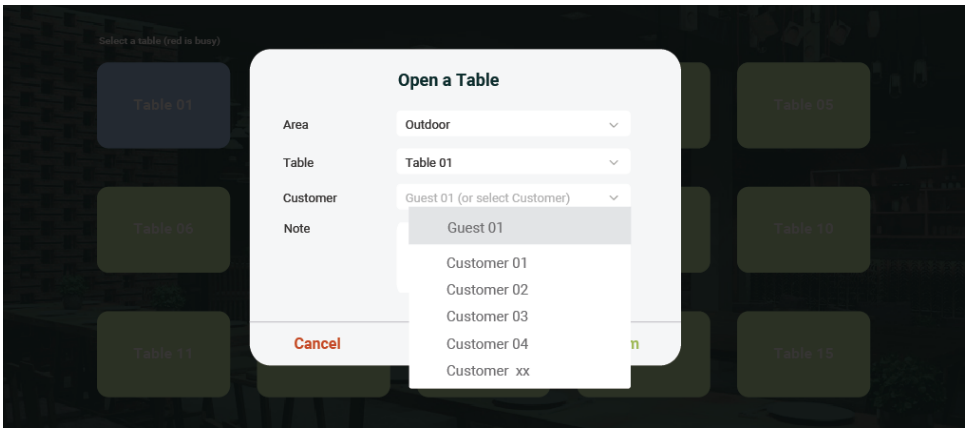


Figura 5.22: Agregar cliente a una mesa abierta.

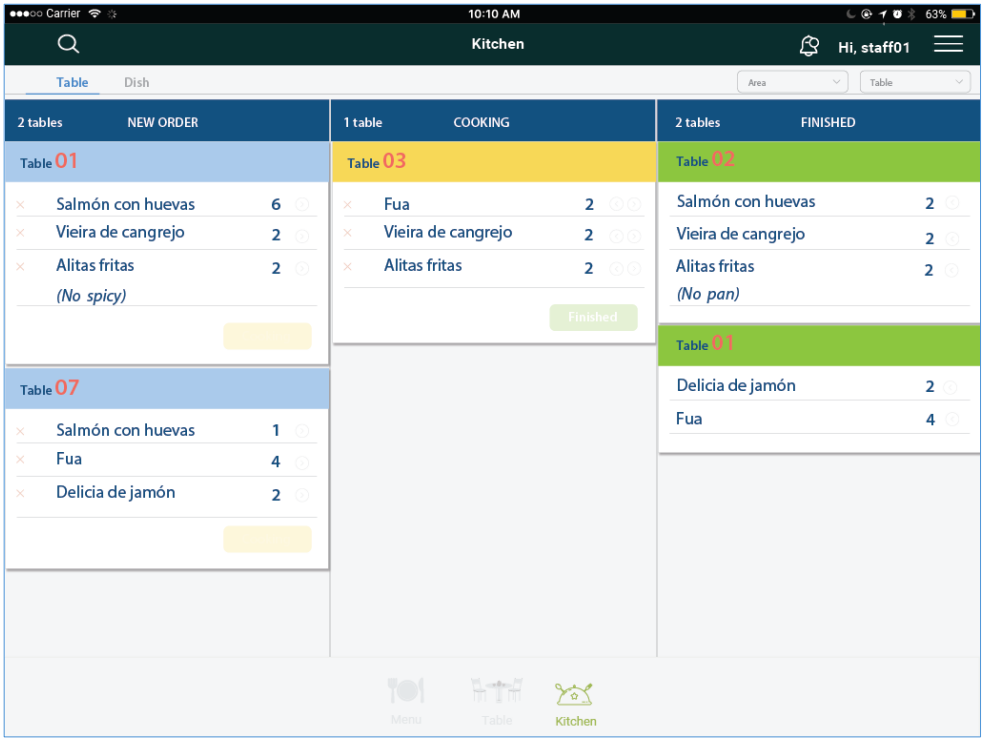


Figura 5.23: Pantalla de cocina con el estado de los platos.

- En el lado derecho de la pantalla (web o tablet) (para teléfonos con pantalla pequeña, desplácese hacia abajo para ver la factura) se mostrará una lista de platos pedidos en esa mesa. Los botones para cancelar pedido, cambiar tabla, cancelar tabla, imprimir factura y pago se pueden encontrar aquí (Figura 5.24).

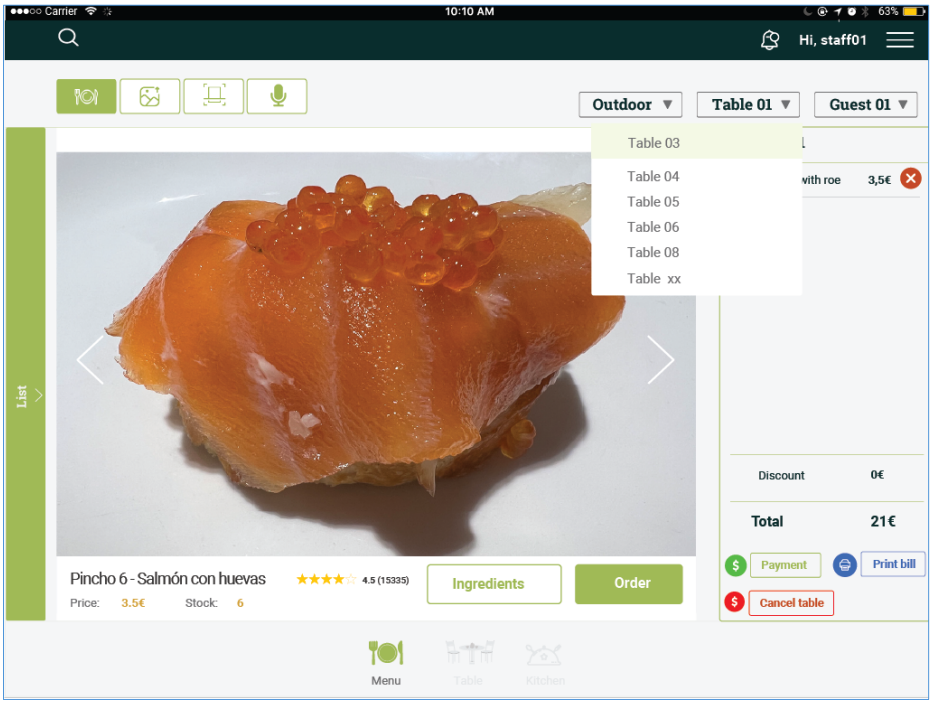


Figura 5.24: Pantalla de menú en la aplicación POS - versión para personal.

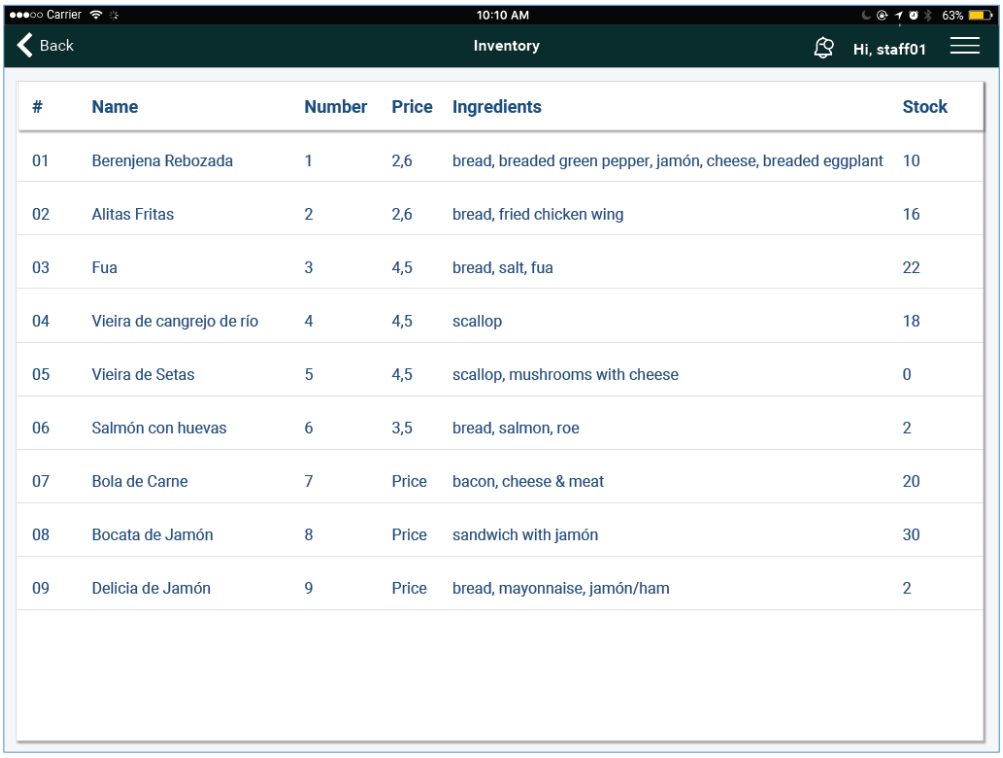
- En la parte superior derecha hay un botón de menú oculto que incluye:

  - Perfil: los empleados pueden actualizar su información como cambiar contraseña, número de teléfono, ...

  - Gestión de inventario: informa rápidamente cuántos artículos quedan en stock.

  - Gestión de clientes: agregar, editar información del cliente, ver el historial de pedidos del cliente, puede reimprimir facturas o reordenar facturas a solicitud del cliente si son clientes leales registrados en el sistema (Figura 5.27).

Figura 5.25: Menú de configuración del personal.



#	Name	Number	Price	Ingredients	Stock
01	Berenjena Rebozada	1	2,6	bread, breaded green pepper, jamón, cheese, breaded eggplant	10
02	Alitas Fritas	2	2,6	bread, fried chicken wing	16
03	Fua	3	4,5	bread, salt, fua	22
04	Vieira de cangrejo de río	4	4,5	scallop	18
05	Vieira de Setas	5	4,5	scallop, mushrooms with cheese	0
06	Salmón con huevas	6	3,5	bread, salmon, roe	2
07	Bola de Carne	7	Price	bacon, cheese & meat	20
08	Bocata de Jamón	8	Price	sandwich with jamón	30
09	Delicia de Jamón	9	Price	bread, mayonnaise, jamón/ham	2

Figura 5.26: Inventorio.

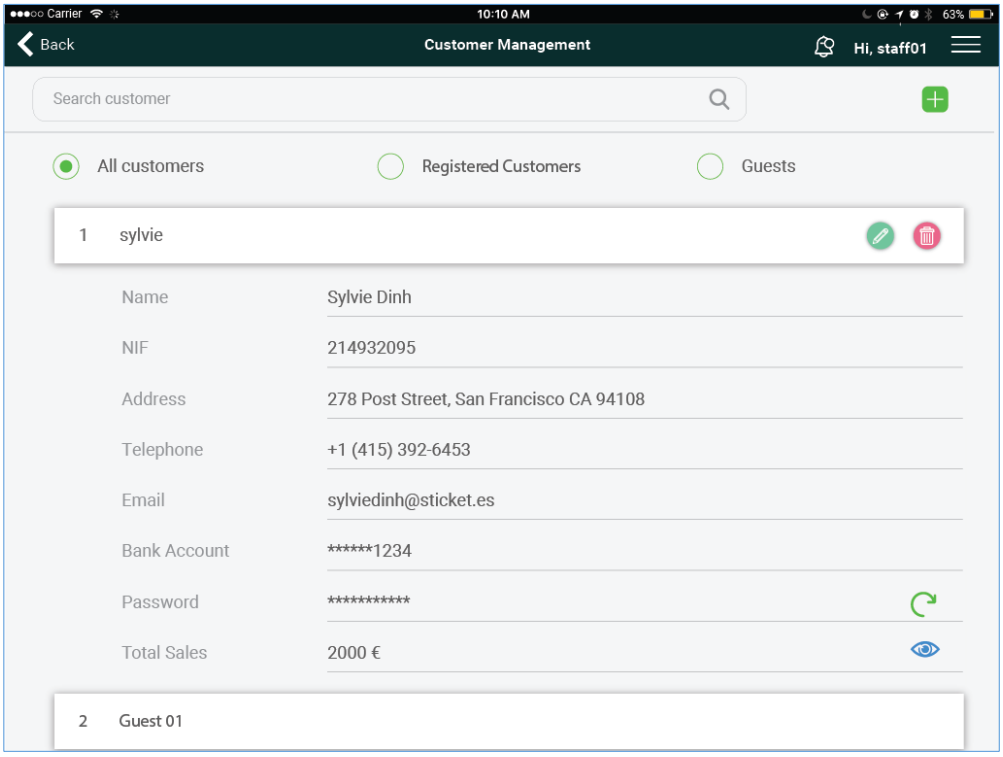


Figura 5.27: Gestión de clientes.

Por otro lado, se otorgan diferentes funciones según cada rol:

Rol de gestión:

El gerente o administrador tiene permiso total para configurar el sistema y establecer permisos para cada usuario, que aquí es el cajero, el camarero, el contador y el chef.

- Gestión de empleados: los gerentes pueden agregar nuevos empleados, editar información y eliminar cuentas de empleados que ya no trabajan en el restaurante.

- Gestión de clientes: al igual que la gestión de empleados, los gerentes pueden agregar clientes, editar información y eliminar cuentas cuando los clientes ya no necesitan utilizar el servicio.

- Administrar sucursales de restaurantes (si hay más de 1 sucursal): Agregar, editar, eliminar sucursal.

- Administrar tablas y áreas de tablas: cambiar área, agregar, editar, eliminar tabla.

- Gestión de artículos: agregar nuevos platos, editar, eliminar, actualizar precios, actualizar inventario.

- Función de pantalla de cocina: los gerentes tienen total autoridad, como el derecho de cancelar platos, actualizar el estado de los platos, como cocinarlos o completarlos como un chef. De hecho, los gerentes a menudo solo ven, pero no utilizan, a menos que el gerente también sea chef.

- Ver sistema de reportes (dependiendo de las necesidades de cada unidad de negocio, el reporte puede sufrir cambios):

    - Resumen de ventas
    - Informe de pedidos
    - Informe de ventas por cliente o por empleado
    - Informe de inventario
    - Informe de descuento
    - Informe de cancelación de pedido
    - Informe de pago
    - Ver los productos más vendidos
    - etc.,

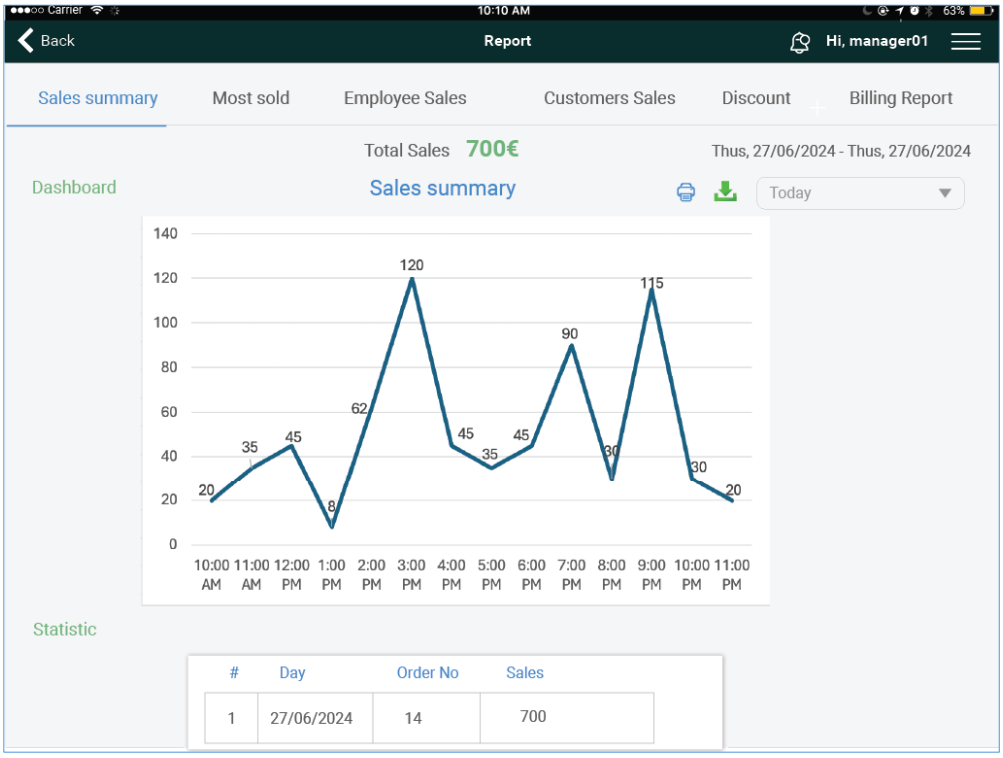


Figura 5.28: Un ejemplo de informe de ventas.

Rol de contable:

- Administrar artículos: agregue nuevos platos, edite, elimine, actualice precios, actualice inventario o cree subcategorías (si corresponde).
- Gestión de inventario: Importar y exportar ingredientes de cocina y actualizar inventario.
- Otras funciones: Imprimir facturas y gestionar pagos.

Rol de camarero:

- Gestión de clientes: agregue clientes, edite información, elimine cuentas cuando los clientes ya no necesiten utilizar el servicio.
- Funciones de pedido (ver Funciones de pedido).
- Pantalla de cocina: solo puede ver el estado de los platos en proceso, recién ordenados o completados, o ver la notificación.

Rol de chef o asistente:

- Función de pantalla de cocina: tiene derecho a cancelar platos, actualizar el estado de los platos procesados o completados. Hay 2 formas de mostrar: Mostrar por mesa (pestaña Tabla) o mostrar por plato (pestaña Plato).

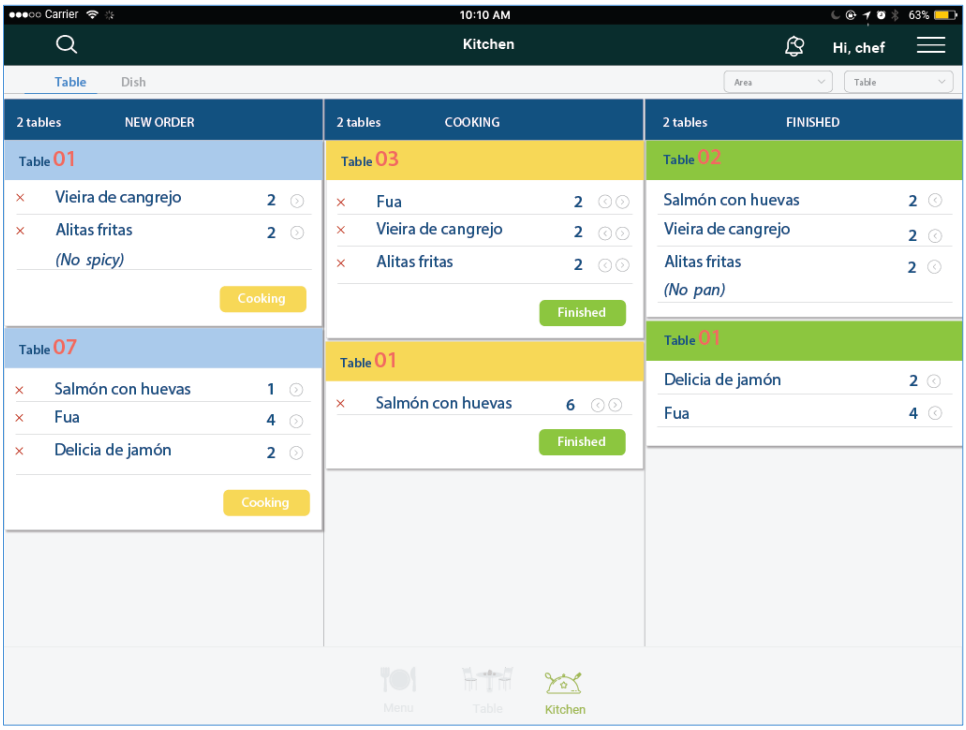


Figura 5.29: Pantalla de cocina – Visualización por mesa.

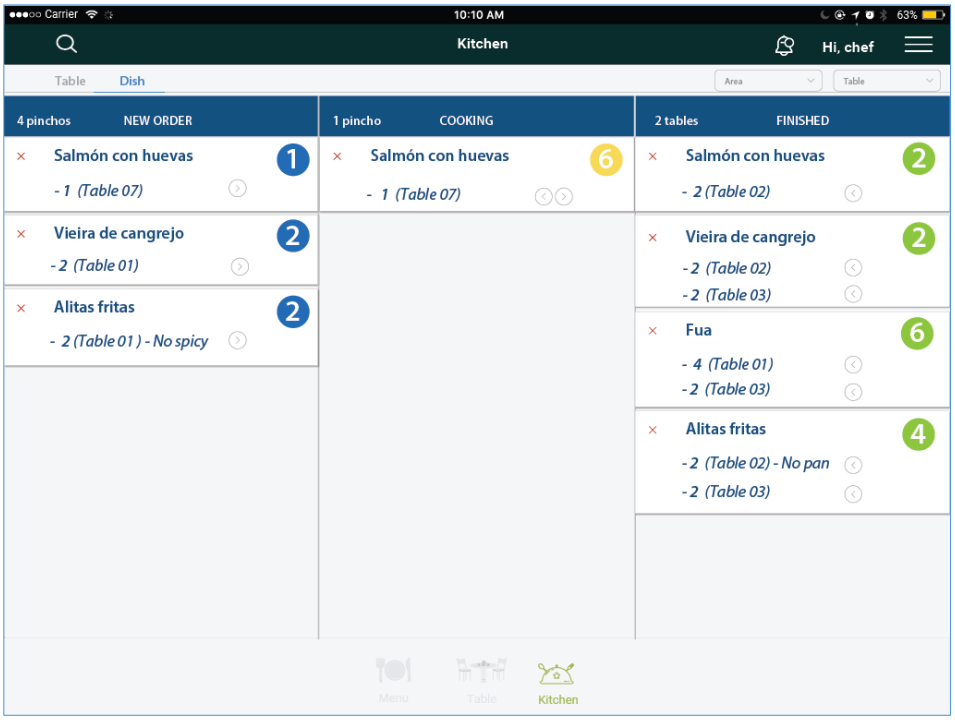


Figura 5.30: Pantalla de cocina – Visualización por plato.

2. Prototipado

Una vez completados, estos diseños se importan a UXPin para crear prototipos y flujos de trabajo interactivos.

UXPin Prototype es una característica que nos permite simular un escenario de flujo de trabajo del sistema POS, probar la usabilidad y recopilar comentarios de las partes interesadas antes de pasar a la etapa de desarrollo del software [27].

Apps	Workflows	Mockups
Versión cliente	1 proceso de pedido (Figura 5.1)	55 pantallas
Versión personal	1 proceso de pedido (Figura 5.2)	75 pantallas

Tabla 5.1: Prototype en UXPin.

El diseño de simulación del sistema POS F&B en versión móvil para clientes y personal está disponible en UXPin Prototype: Prototype de POS F&B Client y Prototype de POS F&B Personal

O se puede ver el flujo de trabajo completo aquí: POS F&B Prototype

Para seguir paso a paso el flujo de trabajo del sistema POS, simplemente haga clic en cada pantalla.

5.2 Colección de datos

El código para implementar esta parte se almacena en Google Drive: Data

1. Preparación de datos

Para entrenar el modelo YOLOv8, el primer paso es preparar los datos, aquí usaré 9 grupos de fotos de 51 fotos que tomé en algunos bares en el verano de 2023 en San Sebastián, España.

Las imágenes originales se almacenan en Google Drive: Original images

Como se muestra en la sección Conceptos teóricos, tabla 3.1, la resolución de la imagen para el entrenamiento en el conjunto de datos de detección de COCO debe ser de 640x640 píxeles.

Para un entrenamiento de alta precisión, estas imágenes también se editan (usando Python) con muchos ángulos diferentes, como rotación, desenfoque y conversión a escala de grises.

```
def rotate_image(image, angle):
    rows, cols = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    rotated_image = cv2.warpAffine(image, rotation_matrix, (cols, rows))
    return rotated_image

def blur_image(image, blur_amount):
    kernel_size = max(1, blur_amount * 2 + 1)
    blurred_image = cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)
    return blurred_image

def resize_image(image, new_width, new_height):
    resized_image = cv2.resize(image, (new_width, new_height))
    return resized_image

def grayscale_image(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Function to process all images with combination:

```
def process_images_in_folder(folder_path, output_folder):
    files = os.listdir(folder_path)
    image_files = [f for f in files if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp',
'.gif'))]

    for file_name in image_files:
        image_path = os.path.join(folder_path, file_name)
```

```
        original_image = cv2.imread(image_path)

        processed_images = []

        for angle in [0, 180]:
            rotated_image = rotate_image(original_image, angle)
            for blur_amount in [0, 5]:
                blurred_image = blur_image(rotated_image, blur_amount)
                for new_width, new_height in [(640, 640)]:
                    resized_image = resize_image(blurred_image, new_width, new_height)
                    processed_images.append((resized_image, angle, blur_amount))

        base_name = os.path.splitext(file_name)[0]
        for i, (processed_image, angle, blur_amount) in enumerate(processed_images):
            processed_name = f"{base_name}_"
            if angle == 180:
                processed_name += "rotated_180_"
            if blur_amount > 0:
                processed_name += "blurred__"
            processed_name += "resized_640_640.jpg"
            cv2.imwrite(os.path.join(output_folder, processed_name), processed_image)

        for i, (processed_image, angle, blur_amount) in enumerate(processed_images):
            grayscaled_image = grayscale_image(processed_image)
            processed_name = f"{base_name}_"
            if angle == 180:
                processed_name += "rotated_180_"
            processed_name += "grayscaled__"
            if blur_amount > 0:
                processed_name += "blurred__"
            processed_name += "resized_640_640.jpg"
            cv2.imwrite(os.path.join(output_folder, processed_name), grayscaled_image)
```

Cada imagen original crea 8 imágenes nuevas con nombres diferentes 'nombre_archivo+combinación', un total de 408 imágenes nuevas:

- blurred__resized__640__640
- grayscaled__blurred__resized__640__640
- grayscaled__resized__640__640
- resized__640__640
- rotated__180__grayscaled__blurred__resized__640__640
- rotated__180__blurred__resized__640__640
- rotated__180__resized__640__640
- rotated__180__grayscaled__resized__640__640.

Procesé todas las imágenes en la carpeta especificada con varias combinaciones de rotación, desenfoque, cambio de tamaño y escala de grises y luego guardé las imágenes procesadas en la carpeta de salida.

```
# folder_path = /Users/sylviedinh/Documents/Projects/ProjectTFM/Data/Images
# output_folder = /Users/sylviedinh/Documents/Projects/ProjectTFM/Data/RetouchImages
folder_path = input("Enter the folder path containing images: ")
output_folder = input("Enter the output folder path for processed images: ")
process_images_in_folder(folder_path, output_folder)
```

- folder_path es la ruta (carpeta) que contiene las 51 imágenes originales que se seleccionaron en el paso anterior.
- output_path es la ruta (carpeta) donde obtenemos las imágenes retocadas del proceso de combinación de imágenes.

Las imágenes retocadas se guardan en el directorio después de ejecutar el comando:
"/Users/sylviedinh/Documents/Projects/ProjectTFM/Data/RetouchImages" para el siguiente paso de preparación de datos y también se cargan en Google Drive con el fin de almacenar los datos para este proyecto: RetouchImages

2. Etiquetado de datos con labelme

Preparar el ambiente:

- Instalar lableme, depende de cada plataforma (Windows, Mac OS o Linux), tenemos diferentes instalaciones específicas [8].
- Entorno virtual activo desde Terminal usando Mac OS en este caso:
  ~ % source path/to/venv/bin/activate
- Ejecutar labelme: (myenv) sylviedinh@Sylvies-MacBook-Pro-M3 ~ % labelme

Los pinchos (de las imágenes retocadas de arriba) están anotados con puntos mediante la creación de polígonos en labelme.

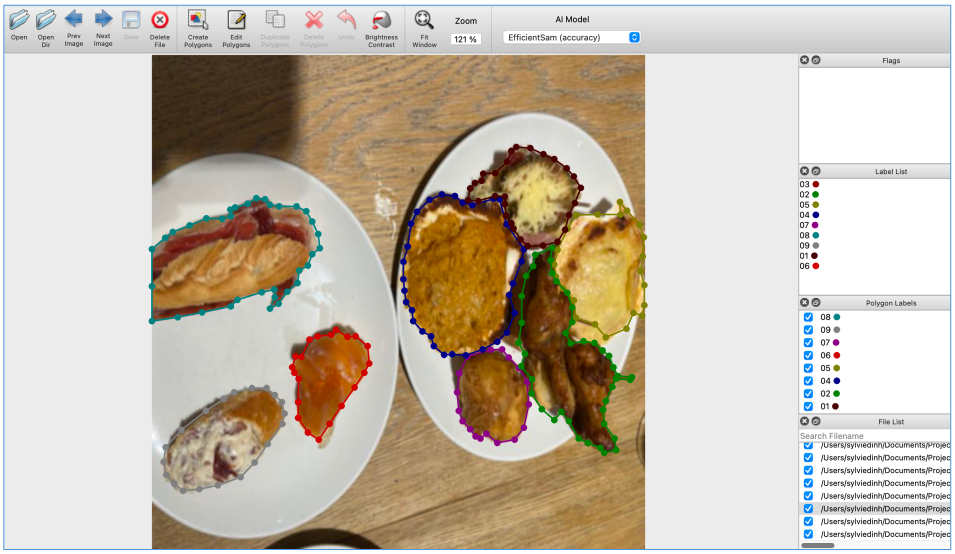


Figura 5.31: Etiqueta una imagen en labelme.

Cada pincho está etiquetado con un número del 1 al 9, según la siguiente lista:

Número	Imagenes ("IMG_* + combinación")[1]
01	1160, 1165, 1198, 1211, 1212, 1213, 1223, 1224, 1226, 1227
02	1132, 1184, 1198, 1217, 1218, 1219, 1223, 1224, 1225, 1226, 1227
03	1116, 1117, 1118, 1120, 1122
04	1145, 1147, 1148, 1208, 1209, 1210
05	1143, 1144, 1147, 1198, 1217, 1220, 1221, 1222, 1223, 1224, 1225
06	1197, 1198, 1199, 1200, 1201, 1223, 1224, 1225, 1227
07	1146, 1198, 1201, 1202, 1214, 1215, 1216, 1217, 1223, 1224, 1226
08	1157, 1161, 1197, 1198, 1205, 1206, 1207
09	1159, 1162, 1163, 1165, 1197, 1198, 1202, 1203, 1204

Tabla 5.2: Lista de imágenes de los pinchos correspondientes a cada número.

[1] Ejemplo, imagen 'IMG_1116' + combinación = "IMG_1116_blurred_resized_640_640.jpg"

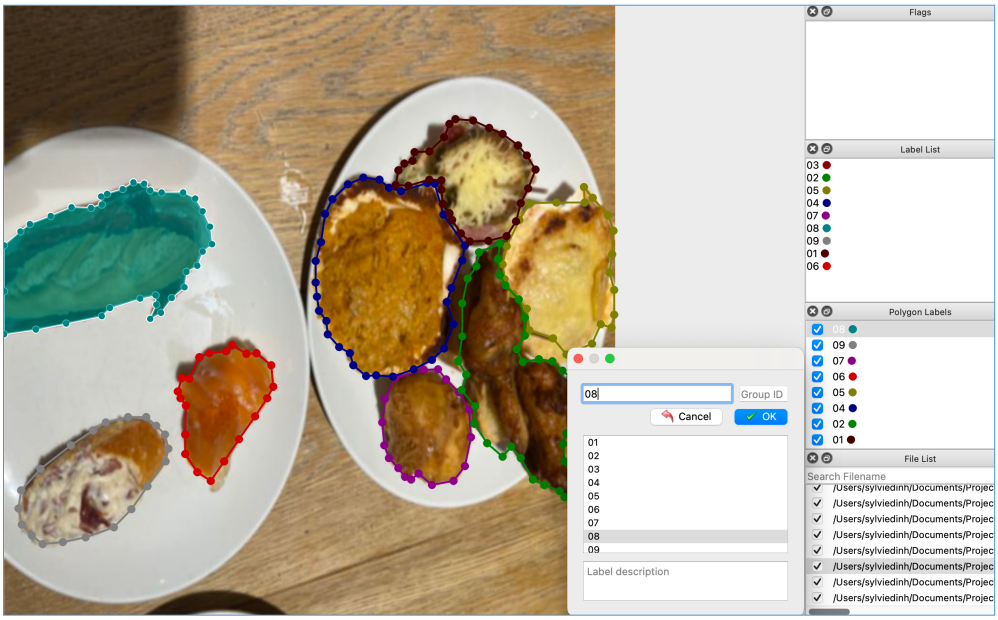


Figura 5.32: Etiquetas de polígonos por ID de grupo en labelme.

Después de etiquetar las imágenes, se guardarán como archivos JSON en la carpeta 'labelme_json_dir'. Un total de 408 archivos en formato JSON, corresponden a 408 imágenes retocadas.

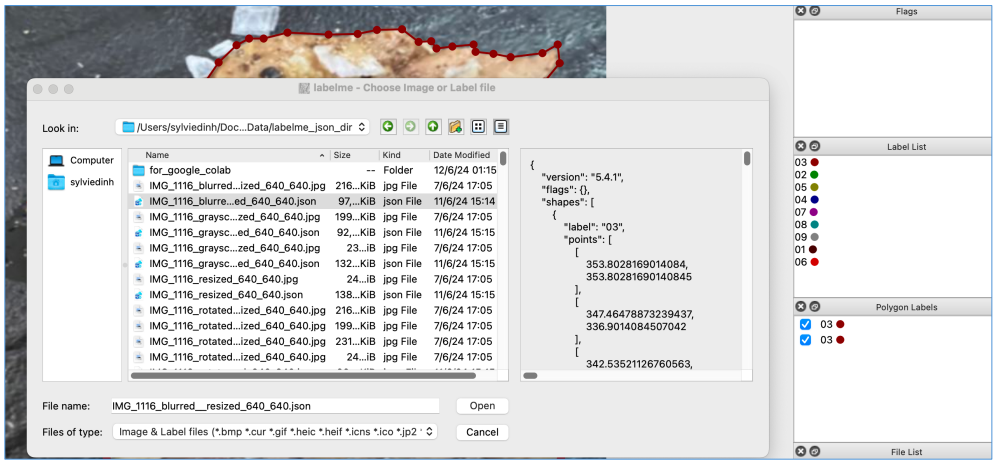


Figura 5.33: Un archivo en formato JSON después del etiquetado

Seleccionar anotaciones con puntos usando la función de polígono para 408 imágenes en labelme en lugar de simplemente crear un cuadro delimitador, lleva mucho tiempo porque tiene que hacer puntos detallados para cada imagen, hay imágenes complejas que consisten en muchos pinchos apilados encima de cada uno. otro. Pero traerá resultados de entrenamiento de mayor precisión para el modelo YOLOv8.

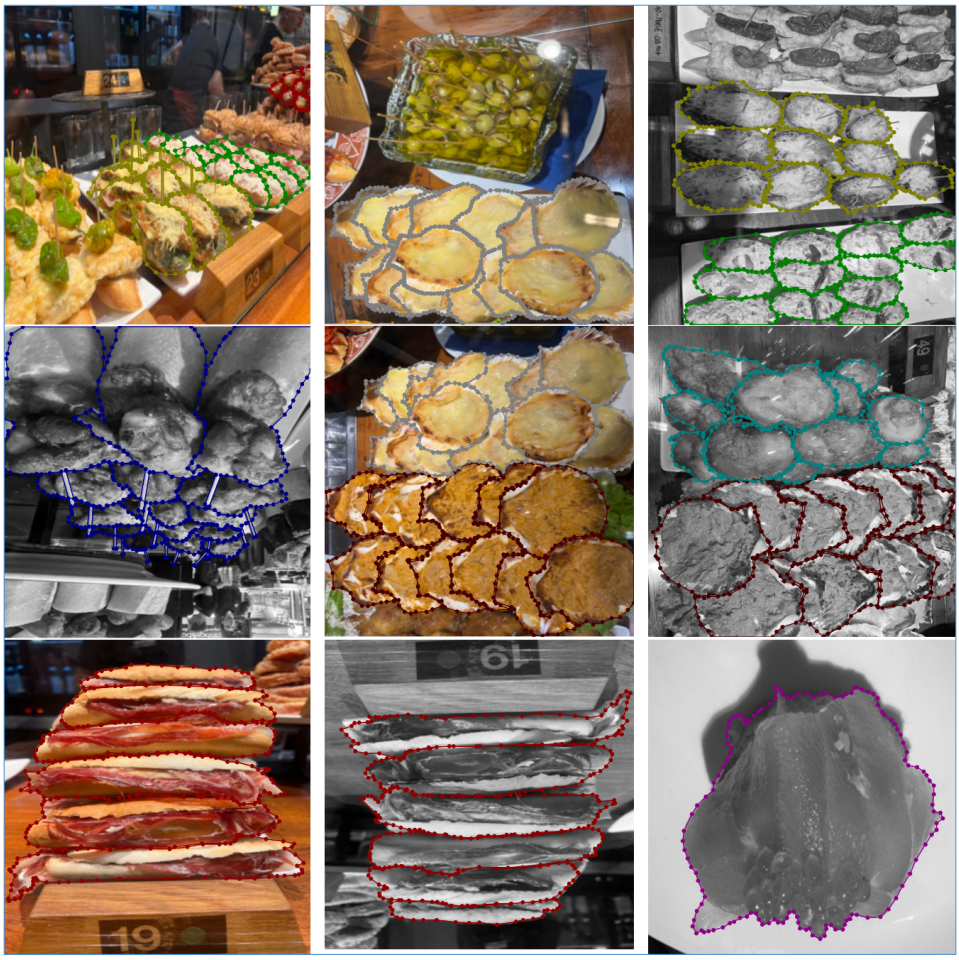


Figura 5.34: Otros ejemplos de etiquetas de imágenes usando LabelMe y conversión de anotaciones al formato JSON.

Las imágenes etiquetadas se almacenan en el directorio: "/Users/sylviedinh/Documents/Projects/ProjectTFM/Data/labelme_json_dir"[1] y subido a Google Drive[2]: labelme_json_dir

3. Crear YOLODataset

Requisitos:

- Instalar labelme2YOLO con "pip install labelme2yolo"
- Importar declaraciones:

```
import os
import shutil
import time
```

Para entrenar el modelo, hay 2 formas: ejecutarlo en local o en la nube (Colab).

[1] Para el siguiente paso "Crear YOLODataset". La ruta local se utiliza para entrenar el modelo en una computadora personal.

[2] La ruta de Google Drive se utiliza para entrenar el modelo en Google Colab.

El siguiente código prepara el conjunto de datos para ejecutarse en ambos entornos. Como se mencionó en la sección de herramientas y tecnologías, entrenar el modelo YOLOv8 requiere una computadora con una configuración potente, y Google Colab con soporte GPU hace que este entrenamiento sea más rápido.

```
def yoloLocal2GoogleColab(path_dir):
    YOLO_LOCAL_DIR_NAME = "YOLODataset"
    GOOGLE_COLAB_DIR_NAME = "for_google_colab"

    if path_dir[-1] != '/' and path_dir[-1] != '\\':
        path_dir += '/'
    print(f"Provided path: {path_dir}")

    if not os.path.exists(path_dir):
        print("ERROR: Directory does not exist.")
        return

    local_dir_base = os.path.join(path_dir, YOLO_LOCAL_DIR_NAME)
    local_dir_train_images = os.path.join(local_dir_base, "images/train/")
    local_dir_val_images = os.path.join(local_dir_base, "images/val/")
    local_dir_test_images = os.path.join(local_dir_base, "images/test/")
    local_dir_train_labels = os.path.join(local_dir_base, "labels/train/")
    local_dir_val_labels = os.path.join(local_dir_base, "labels/val/")
    local_dir_test_labels = os.path.join(local_dir_base, "labels/test/")

    print(f"Checking directories in {local_dir_base}")
    for d in [local_dir_train_images, local_dir_val_images,
local_dir_test_images, local_dir_train_labels, local_dir_val_labels,
local_dir_test_labels]:
        print(f"Checking if {d} exists: {os.path.exists(d)}")

    if not all(os.path.exists(d) for d in [local_dir_train_images,
local_dir_val_images, local_dir_test_images, local_dir_train_labels,
local_dir_val_labels, local_dir_test_labels]):
        print("ERROR: 'labelme2yolo' directory structure does not exist.")
        return

    dir_base = os.path.join(path_dir, GOOGLE_COLAB_DIR_NAME)
    dir_train_name = os.path.join(dir_base, "train/")
    dir_val_name = os.path.join(dir_base, "val/")
    dir_test_name = os.path.join(dir_base, "test/")
    dir_train_images = os.path.join(dir_train_name, "images/")
    dir_train_labels = os.path.join(dir_train_name, "labels/")
    dir_val_images = os.path.join(dir_val_name, "images/")
    dir_val_labels = os.path.join(dir_val_name, "labels/")
    dir_test_images = os.path.join(dir_test_name, "images/")
    dir_test_labels = os.path.join(dir_test_name, "labels/")

    if os.path.exists(dir_base):
```

```
        shutil.rmtree(dir_base)
    time.sleep(1)
    if not os.path.exists(dir_base):
        os.makedirs(dir_train_images)
        os.makedirs(dir_train_labels)
        os.makedirs(dir_val_images)
        os.makedirs(dir_val_labels)
        os.makedirs(dir_test_images)
        os.makedirs(dir_test_labels)

    copyFilesFromTo(local_dir_train_images, dir_train_images)
    copyFilesFromTo(local_dir_val_images, dir_val_images)
    copyFilesFromTo(local_dir_test_images, dir_test_images)
    copyFilesFromTo(local_dir_train_labels, dir_train_labels)
    copyFilesFromTo(local_dir_val_labels, dir_val_labels)
    copyFilesFromTo(local_dir_test_labels, dir_test_labels)

    lines = []
    with open(os.path.join(local_dir_base, "dataset.yaml"), "r") as file_from:
        lines = file_from.readlines()

    yaml_lines = ["path: /content/drive/MyDrive/TFM/\n", "train: train/images\n",
"val: val/images\n", "test: test/images\n"]
    with open(os.path.join(dir_base, "dataset.yaml"), "w") as file_out:
        file_out.writelines(yaml_lines)
        for line in lines:
            if not any(line.startswith(prefix) for prefix in ["path:", "train:",
"val:", "test:"]):
                file_out.write(line)
```

El conjunto de datos se divide en conjuntos de entrenamiento (70%), validación (15%) y prueba (15%).

```
def executeAllForImagesLabelsDir(path):
    os.system(f"labelme2yolo --json_dir {path} --val_size 0.15 --test_size 0.15")
    time.sleep(1)
    yoloLocal2GoogleColab(path)

if __name__ == "__main__":
    path = input("the path of the images directory with their labels: ")
    executeAllForImagesLabelsDir(path)
```

Técnicamente, 'labelme2YOLO' creará dos directorios 'YOLODataset' y 'for_google_colab'[1].

El directorio en dataset.yaml que se ejecuta localmente y en Google Colab es diferente:

[1] 'for_google_colab' es el directorio que va a utilizar para entrenar los datos con el modelo YOLO en Google Colab.

Dataset en local:

```
labelme_jason_dir/
|-- YOLODataset/
|    images/
|      |-- test/
|      |      |-- ***.png
|      |      |-- ...
|      |      |-- ***.png
|      |-- train/
|      |      |-- ***.png
|      |      |-- ...
|      |      |-- ***.png
|      |-- val/
|      |      |-- ***.png
|      |      |-- ...
|      |      |-- ***.png
|    labels/
|      |-- test/
|      |      |-- ***.txt
|      |      |-- ...
|      |      |-- ***.txt
|      |-- train/
|      |      |-- ***.txt
|      |      |-- ...
|      |      |-- ***.txt
|      |-- val/
|      |      |-- ***.txt
|      |      |-- ...
|      |      |-- ***.txt
|    dataset.yaml
|---
```

Dataset ('for_google_colab') para ejecutar en Google Colab:

```
labelme_jason_dir/
|-- for_google_colab/
|    test/
|      |-- images/
|      |      |-- ***.png
|      |      |-- ...
|      |-- labels/
|      |      |      |-- ***.txt
|      |      |      |-- ...
|    train/
|      |-- images/
|      |      |-- ***.png
|      |      |-- ...
|      |-- labels/
|      |      |      |-- ***.txt
|      |      |      |-- ...
|    val/
|      |-- images/
|      |      |-- ***.png
|      |      |-- ...
|      |-- labels/
|      |      |-- ***.txt
|      |      |-- ...
|    dataset.yaml
|---
```

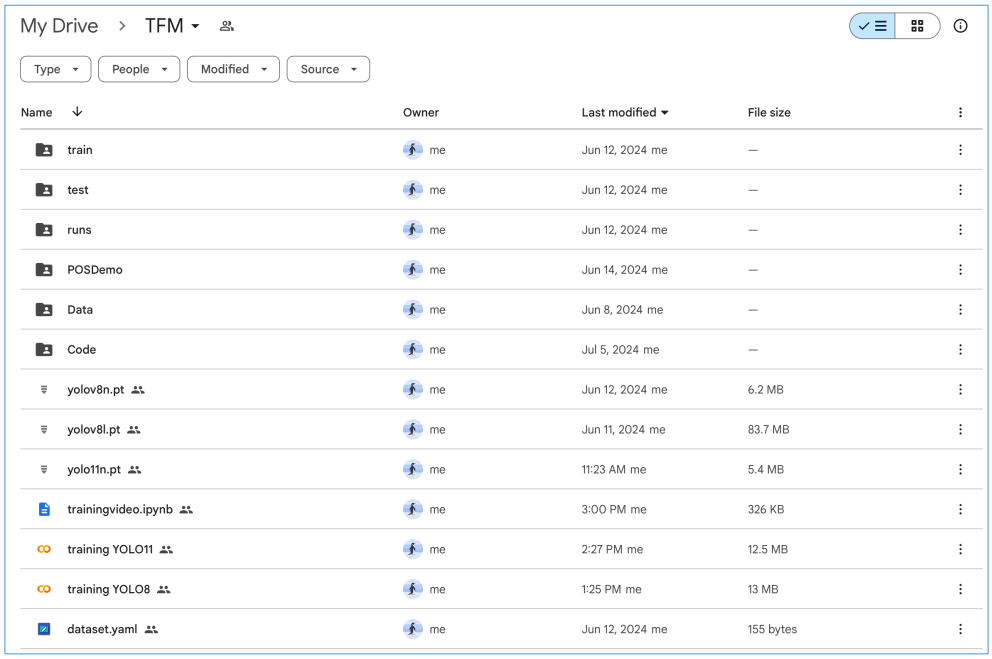


Figura 5.35: Directorio de conjunto de datos en Google Drive.

El archivo 'dataset.yaml' para Google Colab debe ser:

```
path: /content/drive/MyDrive/TFM/
train: train/images
val: val/images
test: test/images
nc: 9
names: ["06", "02", "01", "08", "09", "05", "04", "07", "03"]
```

5.3 Entrenamiento del modelo

Hay dos opciones para entrenar el modelo: entrenamiento en la nube y entrenamiento local [3]:

- El entrenamiento en la nube ofrece una gran escalabilidad y un hardware potente para manejar grandes conjuntos de datos y modelos complejos. Plataformas como Google Cloud, AWS y Azure brindan acceso bajo demanda a GPU y TPU de alto rendimiento, lo que acelera los tiempos de capacitación y permite experimentar con modelos más grandes. Sin embargo, el uso a largo plazo es costoso y la transmisión de datos puede agregar costos y latencia.

- El entrenamiento local proporciona un mayor control, lo que nos permite adaptar el entorno a nuestras necesidades y es más seguro. Sin embargo, el hardware local puede tener limitaciones de recursos y requisitos de

mantenimiento, lo que puede llevar a tiempos de capacitación más prolongados para modelos grandes.

El entrenamiento del modelo YOLOv8 para este proyecto se implementó en Google Colab.

Como se muestra en la Figura 5.35, después de lograr el conjunto de datos, se descargua el archivo de configuración del modelo 'yolo8l.pt' [6] (modelo grande pre-entrenado YOLOv8) y luego se cagan todos en Google Drive.

Preparar entorno:

- Cree un nuevo cuaderno en Google Colab, con nombre 'training YOLO8.ipynb'.
- Vaya a Tiempo de ejecución\Cambiar tipo de tiempo de ejecución, elija T4GPU.
- Para mostrar el estado de la GPU NVIDIA, incluido el uso de la memoria y el uso de la GPU: !nvidia-smi
- Installación: ultralytics
- Montaje de Google Drive (mounting):

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/TFM
```

- El código completo del proceso de entrenamiento y la visualización de resultados se guardan en Google Colab: training YOLO8.ipynb

1. Entrenamiento del modelo

Se entrena el modelo YOLOv8 en el conjunto de datos con 300 épocas de entrenamiento utilizando el archivo de configuración del modelo yolo8l.pt y establezca el tamaño de la imagen en 640x640 píxeles. También se crean gráficos de entrenamiento (E.g. gráfico normalizado de matriz de confusión – Figura 5.40, gráfico de correlograma de etiquetas – Figura 5.41, gráfico de distribución de etiquetas – Figura 5.42 y gráfico de curva de confianza F1 – Figura 5.43).

```
!yolo task=detect mode=train model=yolov8l.pt data= dataset.yaml epochs=300
imgsz=640 plots=True
```

Una vez entrenado el modelo, YOLO genera automáticamente el directorio 'ejecuciones' y se ponderará en la ruta 'run/train/weights'. El mejor peso se

guardará como "best.pt", este modelo se utiliza para obtener nuevas predicciones de imágenes.

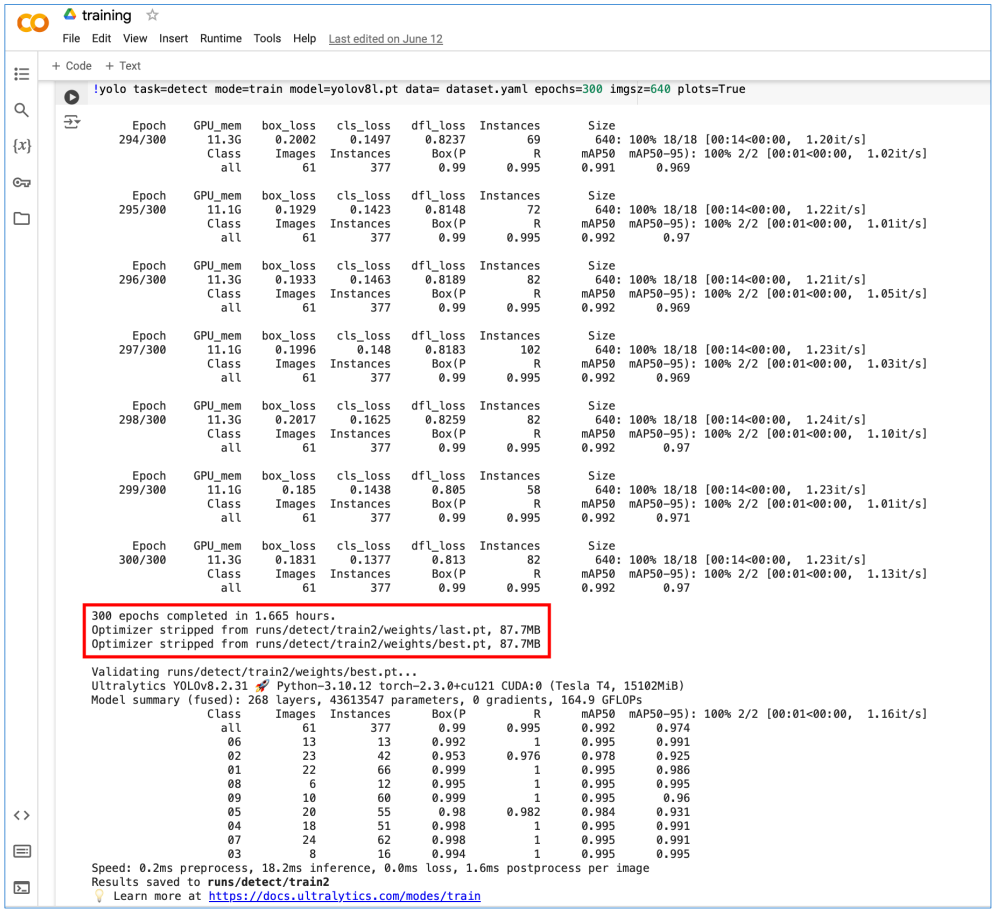


Figura 5.36: Entrenamiento del modelo YOLOv8 en Google Colab.

El archivo 'last.pt' se utilizará si continuamos entrenando nuevos datos.

2. Análisis de resultados

Como se muestra en la Figura 5.36, los resultados se generan durante el entrenamiento y se guardan en run/detect/train2 en Google Drive: runs/detect/train2

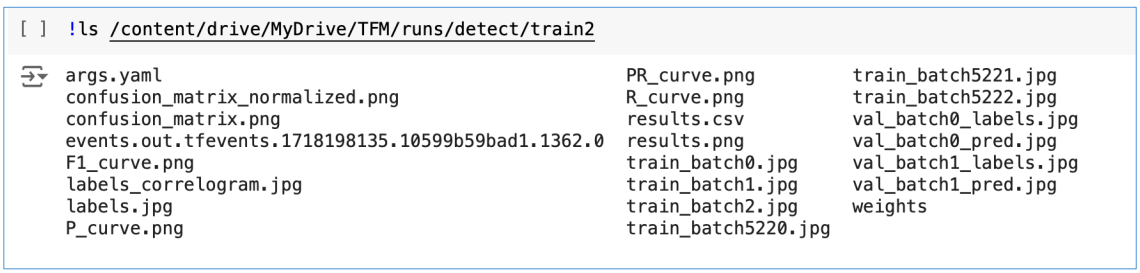


Figura 5.37: Resultados obtenidos tras el entrenamiento con YOLOv8.

Los resultados muestran varias visualizaciones con gráficos de matriz de confusión, gráficos de correlación, curvas de rendimiento, imágenes de lotes de validación y entrenamiento.

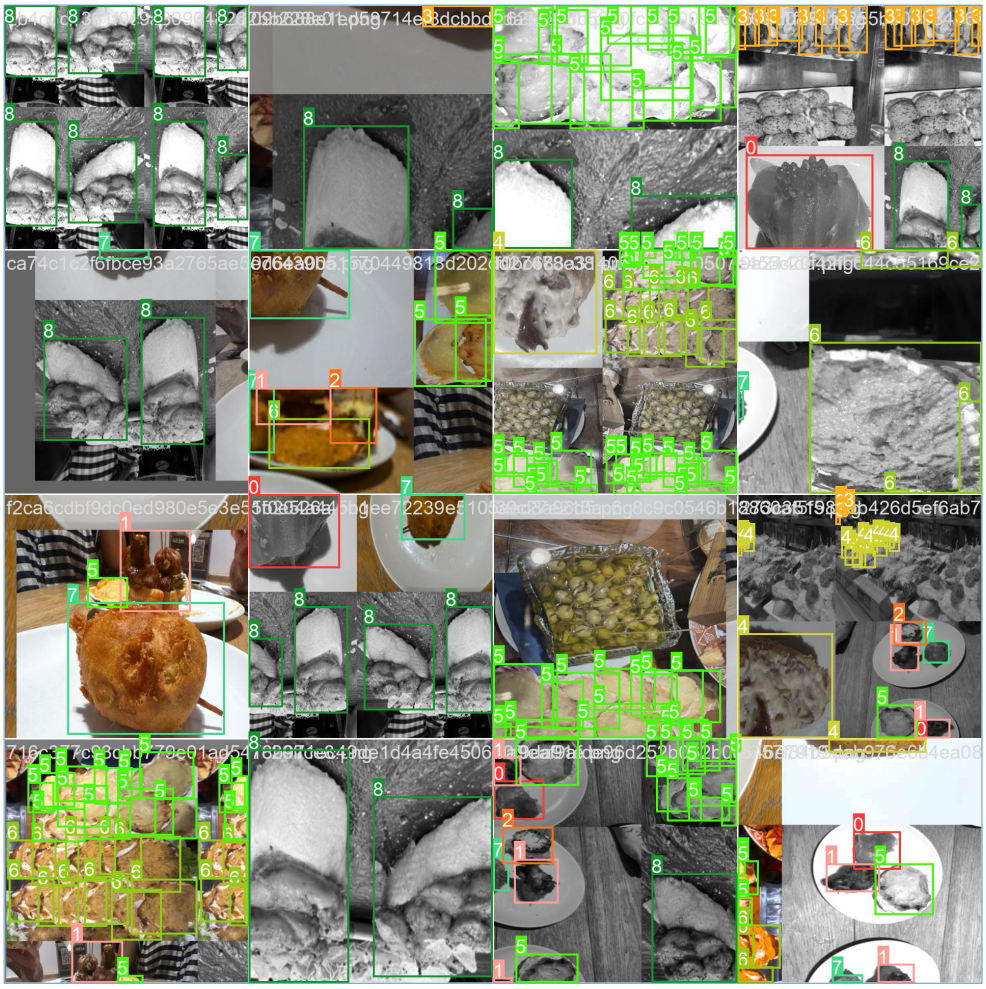


Figura 5.38: Visualización de lotes de entrenamiento de YOLOv8.

La figura 5.38 muestra el resultado en el lote de entrenamiento de que el modelo vio estas imágenes y ajustó sus pesos para mejorar la precisión de la detección. La densidad de los cuadros delimitadores indica un conjunto de datos de entrenamiento rico, con la posibilidad de tener más instancias del mismo tipo en una imagen, algunas de las cuales tienen cuadros delimitadores.

La figura 5.39 a continuación también muestra varias imágenes con múltiples cuadros delimitadores y etiquetas con diferentes números (se han agrupado en la etiqueta para cada pincho numerados del 1 al 9). El conjunto de datos de validación es un conjunto separado de imágenes que el modelo no vio durante el entrenamiento. El lote de validación tiene como objetivo evaluar el rendimiento y garantizar que el modelo se generalice bien a datos invisibles.

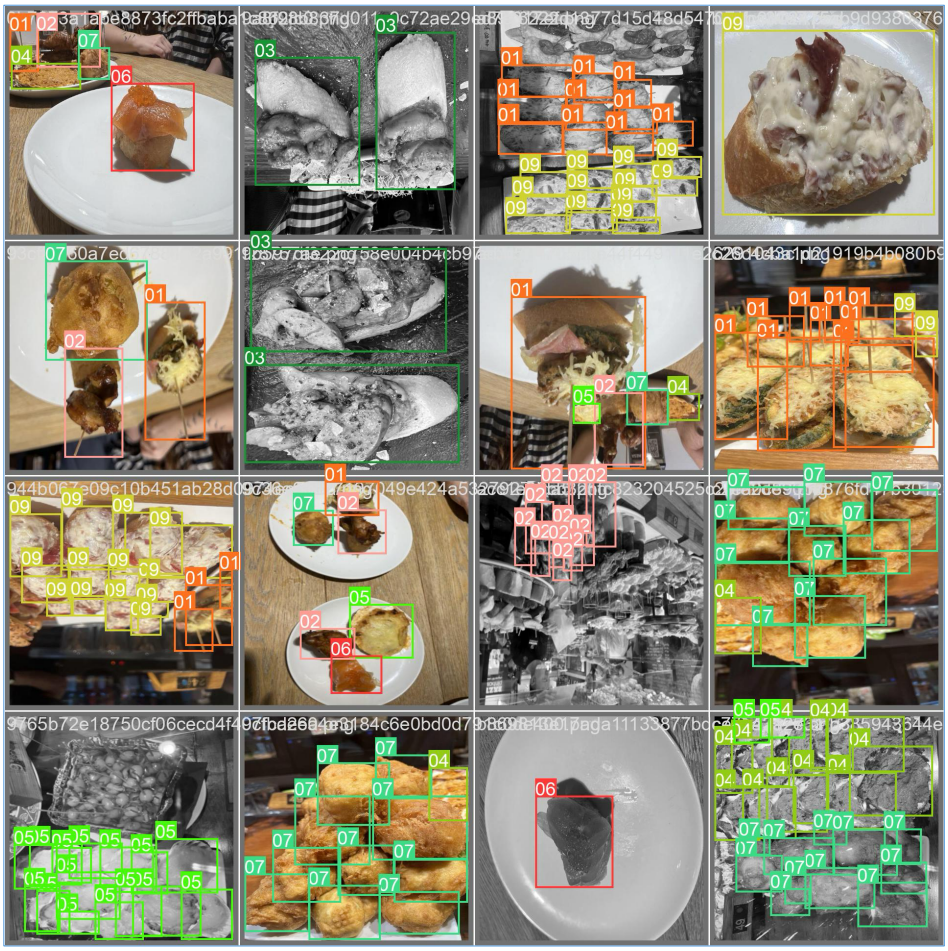


Figura 5.39: Visualización de lotes de validación de YOLOv8.

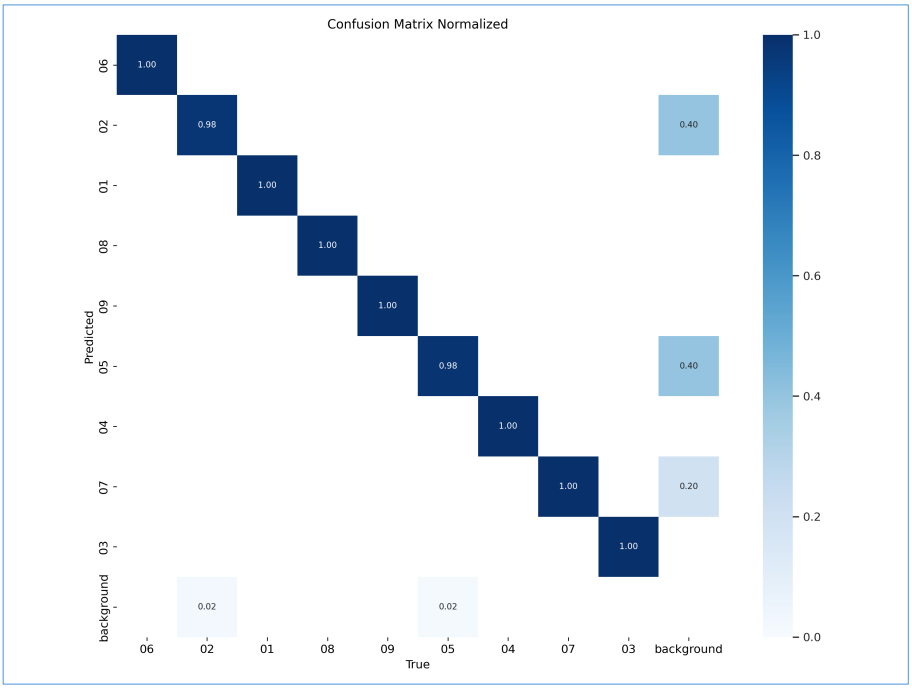


Figura 5.40: Gráfico normalizado de matriz de confusión.

El gráfico Normalizado de la Matriz de Confusión (Figura 5.40) muestra la proporción de predicciones correctas e incorrectas en relación con el número total de predicciones. La línea diagonal muestra el porcentaje de instancias clasificadas correctamente para cada clase (las clases o pinchos están numerados del 1 al 9). Una versión de clase de 1.00 significa que el modelo tiene una clasificación 100% precisa. El pincho número 2 y el pincho número 5 muestran un 2% de casos clasificados incorrectamente como otra clase. Además, ambos tienen una clasificación errónea de antecedentes de 0,40 o el 40% de los casos deberían clasificarse como pincho en lugar de clasificarse como fondo.

Correlograma de etiquetas (Labels Correlogram): visualiza la correlación entre diferentes etiquetas (clases), identifica dependencias o co-ocurrencias entre clases.

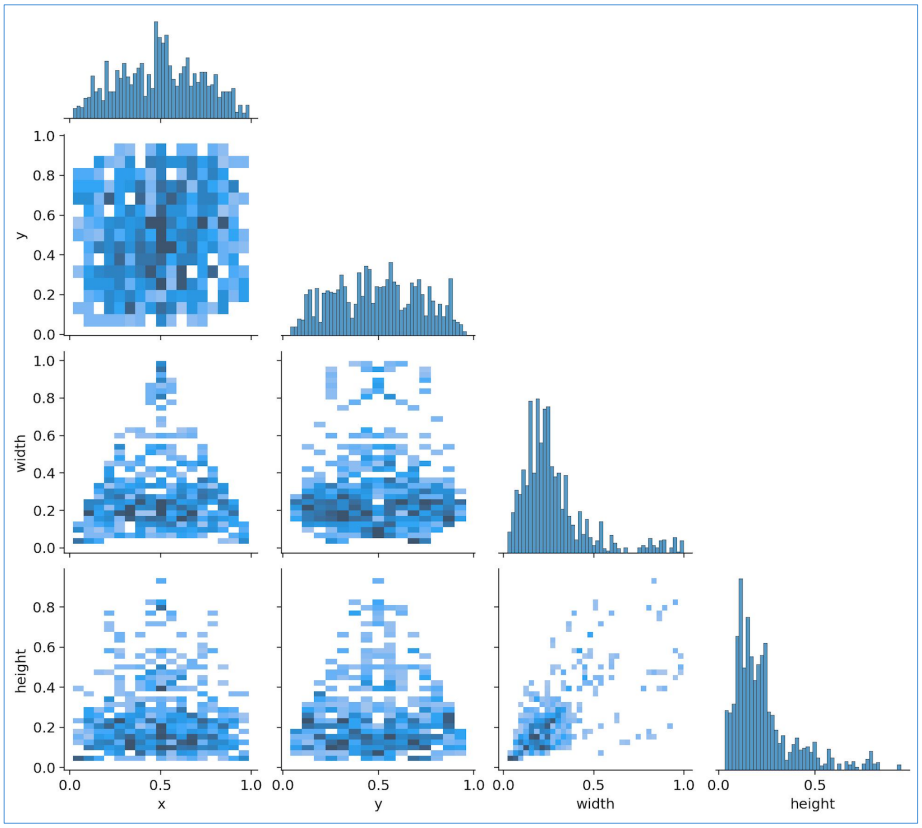


Figura 5.41: Gráfico de correlograma de etiquetas.

La Figura 5.41 anterior muestra dos tipos de gráficos:

- Histograma (Diagonal): muestra la distribución de un único atributo (x corresponde al ancho e y corresponde al alto) de los cuadros delimitadores. Este gráfico muestra que la distribución de las coordenadas centrales de los cuadros delimitadores es relativamente uniforme.

- Diagrama de dispersión/ Scatter plot (triángulo superior): muestra la relación entre el ancho (x) y el alto (y) de los cuadros delimitadores, lo que representa la relación de aspecto de los objetos detectados. Distribución de etiquetas Representación visual de la distribución de etiquetas en el conjunto de datos. Muestra con qué frecuencia aparece cada clase en el conjunto de datos.

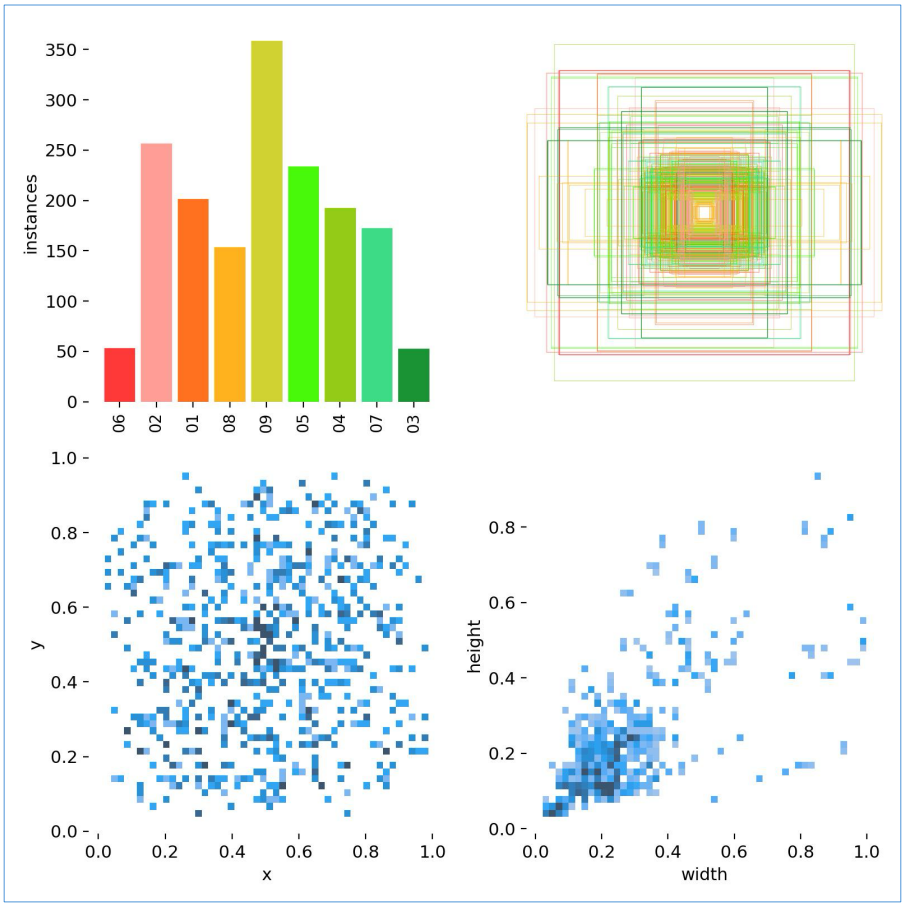


Figura 5.42: Gráfico de distribución de etiquetas.

Gráfico de distribución de etiquetas (Labels distribution plot) representa visualmente la distribución de etiquetas en el conjunto de datos. Muestra con qué frecuencia aparece cada clase en el conjunto de datos.

- Arriba a la izquierda (gráfico de barras): muestra el número de instancias de cada pincho en el conjunto de datos. El Pincho número 9 tiene la mayor cantidad de versiones.
- Arriba a la derecha (cuadros delimitadores superpuestos/ overlapping bounding boxes): muestra la superposición de cuadros delimitadores para todos los pinchos. Parece que el modelo de detección tendrá más

dificultades para detectar los grupos 6, 8 y 7 porque la superposición de estos grupos es mayor.

- Abajo a la izquierda (diagrama de dispersión/ scatter plot): al representar la distribución del centroide de los cuadros delimitadores de la imagen, los objetos aparecen relativamente uniformes en toda la imagen.

- Abajo a la derecha (gráfico de dispersión de la altura (y) y el ancho (x)): muestra la relación entre la altura y el ancho de los cuadros delimitadores. Los puntos cerca de la diagonal representan objetos con una relación de aspecto de 1:1.

La curva F1 traza la puntuación F1 (promedio armónico de precisión y recuperación) bajo diferentes umbrales de confianza. Muestra que la puntuación F1 más alta alcanzada (0,99) se produce en un umbral de confianza de 0,885.

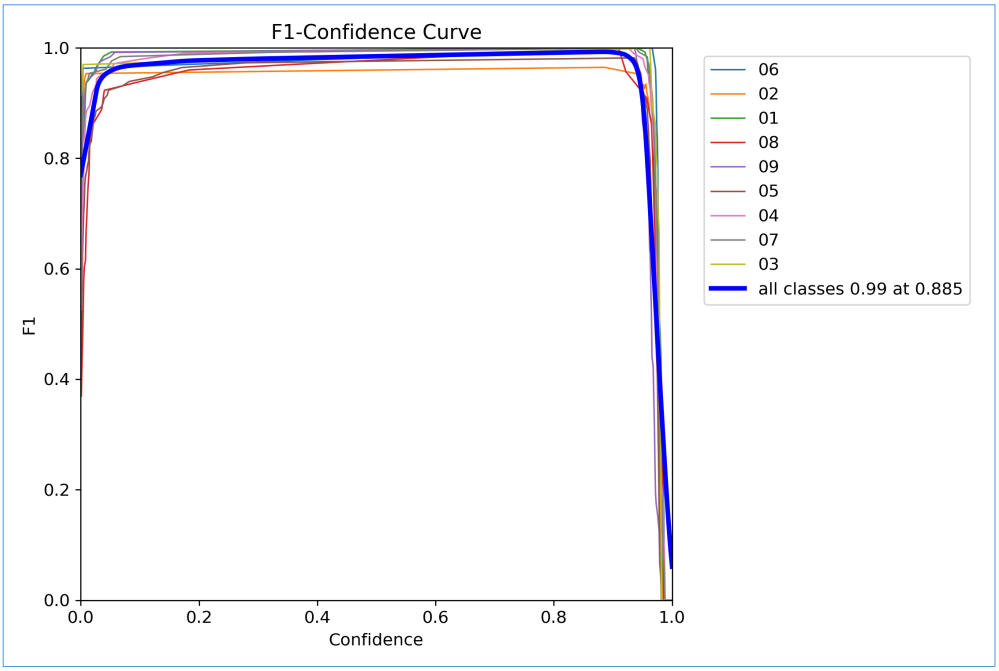


Figura 5.43: Gráfico de curva de confianza F1.

3. Predicción

Utilizamos el modelo entrenado 'best.pt' (`/runs/detect/train2/weights/`) para hacer predicciones sobre imágenes específicas.

```
!yolo task=detect mode=predict
model="/content/drive/MyDrive/TFM/runs/detect/train2/weights/best.pt"
source="/content/drive/MyDrive/TFM/runs/detect/train2/train_batch1.jpg"
```

Los resultados se guardan en runs/detect/predict3 en Google Drive: runs/detect/predict3

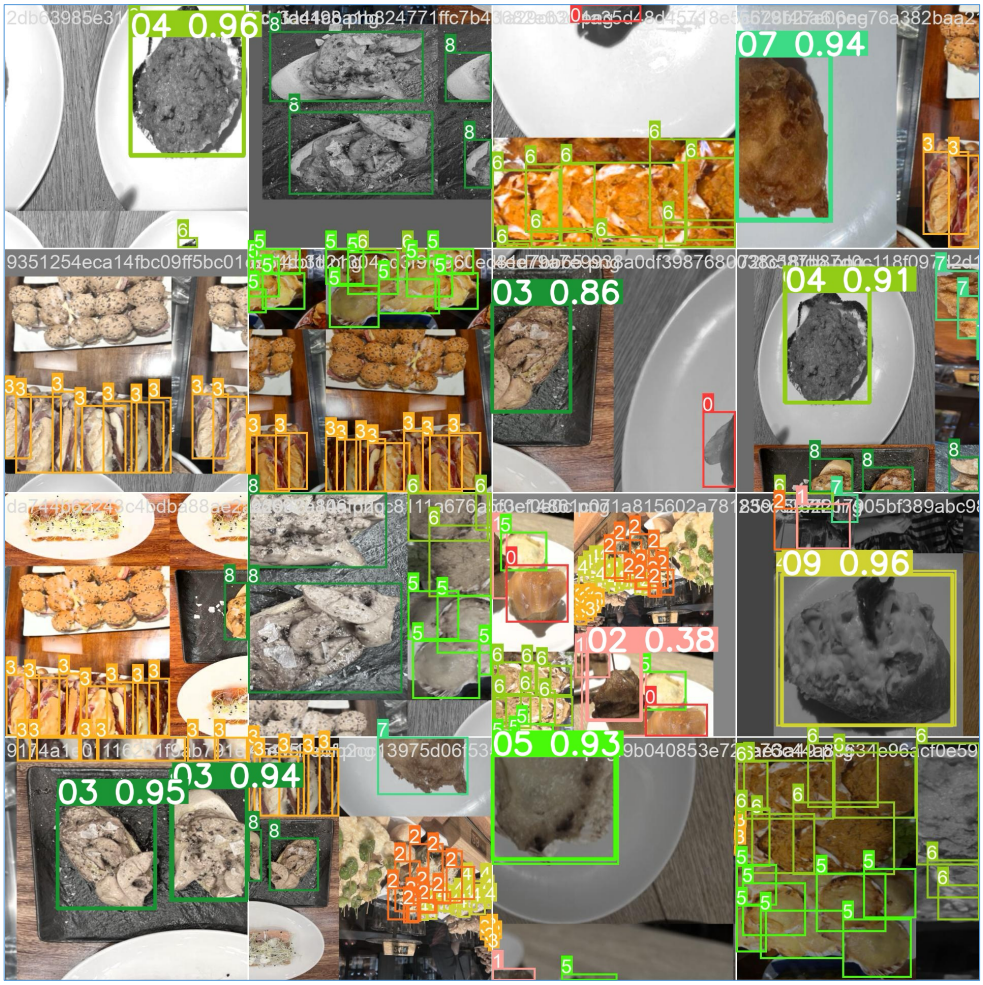


Figure 5.44: Predicción de pinchos.

El pincho número 9 y el pincho número 6 fueron identificados con una precisión casi extremadamente alta del 96%. El pincho número 2 parece predecir con un porcentaje pequeño pero porque el ángulo de visión de la imagen es bastante pequeño y poco claro. (Probé con otras imágenes de este pincho en la demo de POS y el sistema de reconocimiento no tuvo ningún problema).

En general, los resultados del entrenamiento con el modelo YOLOv8 indican una alta precisión de detección y predicción.

4. Aplicar en la prática

El modelo 'best.pt' obtenido del proceso de entrenamiento representa los pesos óptimos para el modelo YOLOv8 según el conjunto de datos de acuerdo con los pasos realizados en las sesiones anteriores.

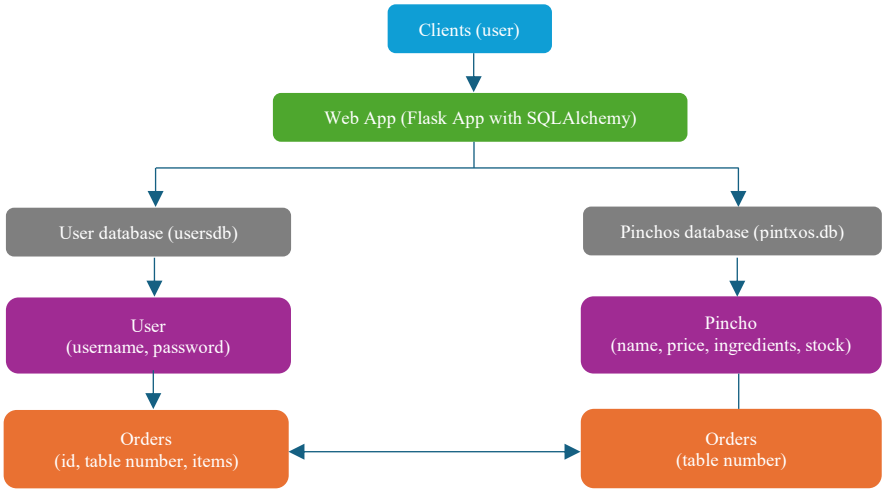
Este modelo se integrará en la demostración del sistema POS para detectar artículos específicos (los pinchos) mediante la carga o escaneo de imágenes desde la aplicación POS.

5.4 POS de demostración y recomendador

Se creó una demostración de POS para demostrar la idea de desarrollar un sistema POS que se diferencia de los sistemas POS convencionales mediante el uso de tecnología de reconocimiento de objetos con aprendizaje automático. De esta manera, los clientes pueden experimentar el pedido de comida escaneando o cargando imágenes en la aplicación, el sistema identificará el plato inmediatamente en tiempo real.

1. Modelos y Base de datos

La aplicación de demostración Flask Web POS que utiliza SQLAlchemy está diseñada para interactuar con dos bases de datos: pintxos.db para el elemento del menú y user.db para la autenticación del usuario, para facilitar la función de pedido.



- Clients: Los usuarios interactúan con la aplicación web.
- Web App (Flask Aplicación): La principal aplicación de realizar pedidos.
- Users: son los que inician sesión usando las credenciales almacenadas en users.db.
- Pintxos: Los pinchos con atributos como número, nombre, precio, ingredientes y existencias se almacenan en pintxos.db.

- Orders: Los usuarios crean pedidos, seleccionando elementos (pinchos) del menú, almacenados en pintxos.db.

Esquema de base de datos

Dimensión	Descripción	Tipos de datos
id	Identificador único del pincho	Integer, Primary key
number	Pincho numerado único	Integer
name	Nombre del pincho	String(100)
price	El precio del pincho	Float
ingredients	Ingredientes que contiene cada pincho	String(200)
allergens	Ingredientes podrían causar algunas reacciones alérgicas.	String(255)
stock	Cantidad de pincho que queda en stock	Integer
image_url	URL de la imagen del pincho	String(200)
notes	Notas	String(500)

Tabla 5.3: Dimensiones de datos de pincho.

Dimensión	Descripción	Tipos de datos
id	Identificador único para la mesa	Integer, Primary key
number	Mesa numerada única	Integer
capacity	Capacidad de cada mesa (optional)	Integer

Tabla 5.4: Dimensiones de datos de la mesa.

Dimensión	Descripción	Tipos de datos
id	Identificador único para cada pincho del pedido.	Integer, Primary key
order_id	Conecta el ítem/pincho a tabla Order.	Foreign key
name	Nombre del pincho pedido.	String (100)
quantity	Cantidad del pincho pedido.	Integer
note	Nota adicional para el pincho.	String (500)
image_url	Ruta a la imagen del pincho.	String (200)
status	Estado de los pinchos (pendiente, en proceso, completado).	String (20)

Tabla 5.5: Dimensiones de datos del pedido.

Dimensión	Descripción	Tipos de datos
id	Identificador único del pedido.	Integer, Primary key
table_number	Número de la mesa donde se realiza el pedido	Integer
items	Objeto JSON que contiene el artículo del pedido	JSON
order_item	Conecta este pedido a múltiples entradas de OrderItem	Relationship

Tabla 5.6: Dimensiones de datos del pedido por cada mesa.

Dimensión	Descripción	Tipos de datos
id	Identificador único para el usuario	Integer, Primary key
username	Nombre de usuario único para autenticación	String(80)
password	Contraseña hash para autenticación	String(200)

Tabla 5.7: Dimensiones de datos del usuario.

Dimensión	Descripción	Tipos de datos
id	Identificador único para cada registro del historial de pedidos.	Integer, Primary key
username	Nombre del usuario que realize el pedido.	String (100)
table_number	El número de mesa asociado con el pedido.	Integer
order	Almacena todos los detalles del pedido.	JSON
total	El precio total del pedido.	Float
tamstamp	La hora en la que se realizó el pedido	Datetime

Tabla 5.8: Historial de los pedidos completados o pasados.

Dimensión	Descripción	Tipos de datos
id	Identificador único para cada registro del historial de pedidos.	Integer, Primary key
username	Nombre del usuario que realize el pedido.	String (100)
table_number	El número de mesa asociado con el pedido.	Integer
order	Almacena todos los detalles del pedido.	JSON
tamstamp	La hora en la que se realizó el pedido	Datetime
status	Estado de los pinchos (pendiente, en proceso, completado).	String (20)

Tabla 5.9: Registro de pedidos de cocina.

Las siguientes clases en “models.py” definen la estructura de las tablas Pintxo, Tabla, Orden, Usuario usando SQLAlchemy ORM (Object-Relational Mapping):

```
class Pintxo(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True, nullable=False)
    number = db.Column(db.Integer, unique=True, nullable=False)
    price = db.Column(db.Float, nullable=False)
    ingredients = db.Column(db.String(200), nullable=False)
    allergens = db.Column(db.String(255), nullable=True)
    stock = db.Column(db.Integer, nullable=False)
    image_url = db.Column(db.String(200))
    notes = db.Column(db.String(500), nullable=True)

    def __repr__(self):
        return f'<Pintxo {self.name}>'


class TableOrder(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    number = db.Column(db.Integer, nullable=False, unique=True)
    capacity = db.Column(db.Integer, nullable=False)
    is_available = db.Column(db.Boolean, default=True)

    def __repr__(self):
        return f'<Table {self.number}>'


class OrderItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    order_id = db.Column(db.Integer, db.ForeignKey('orders.id'), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    quantity = db.Column(db.Integer, nullable=False)
    note = db.Column(db.String(500), nullable=True)
    image_url = db.Column(db.String(200))
    status = db.Column(db.String(20), default='pending')

    def __repr__(self):
        return f'<OrderItem {self.name} - Quantity {self.quantity}>'


class Orders(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    table_number = db.Column(db.Integer, nullable=False)
    items = db.Column(db.JSON, nullable=False)
    order_items = db.relationship('OrderItem', backref='order', lazy=True)

    def __repr__(self):
        return f'<Order {self.id} - Table {self.table_number}>'
```

```
class User(db.Model):
    __bind_key__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)

    def __repr__(self):
        return f'<User {self.username}>'

    def delete(self):
        db.session.delete(self)
        db.session.commit()


class OrderHistory(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), nullable=False, default="Guest")
    table_number = db.Column(db.Integer, nullable=False)
    orders = db.Column(db.JSON, nullable=False)
    total = db.Column(db.Float, nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow())

    def __repr__(self):
        return f'<OrderHistory {self.id} - {self.username}>'


class KitchenDisplay(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), nullable=False, default="Guest")
    table_number = db.Column(db.Integer, nullable=False)
    orders = db.Column(db.JSON, nullable=False)
    total = db.Column(db.Float, nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow())
    status = db.Column(db.String(20), default='pending')

    def __repr__(self):
        return f'<KitchenDisplay {self.id} - {self.username}>'
```

2. Inicialización de la aplicación web

Se divide en dos partes: Front-end y Back-end.

Front-end

El front-end es responsable de mostrar la interfaz de usuario e interactuar con los usuarios. Las tecnologías involucradas para mostrar la interfaz de usuario e interactuar con los usuarios son HTML, CSS y JavaScript. Las plantillas HTML son renderizadas por Flask.

Estructura	Fichero	Descripción	Figura
Templates	index.html	Página de inicio.	Figura 5.45 Figura 5.46
	register.html	Página de registro de usuario.	Figura 5.47
	login.html	Página de inicio de sesión de usuario.	
	profile.html	Página de perfil de usuario (ver, editar información de usuario y eliminar cuenta).	Figura 5.48
	edit_profile.html	Editar página de perfil.	
	menu.html	Página de menú con muestras de 9 pinchos en base de datos.	Figura 5.49
	upload.html	Subiendo páginas de imágenes de pincho.	
	scan.html	Página para escanear elementos usando la cámara.	
	voice.html	Página para realizar pedidos por voz (Diga un número del 1 al 9).	
	confirm_table.htm	Página de confirmación de la tabla seleccionada.	
	order_confirmation .html	Página de mostración de detalles del pedido final.	Figura 5.50
	order_history.html	Página de historial de los pedidos por cada usuario.	
	result.html	Devuelve resultados de cargar o escanear imágenes según el modelo de Yolo para detectar.	
	inventory.html	Página de inventario.	Figura 5.51
	kitchen.html	Página de pantalla de cocina.	Figura 5.52
	dashboard.html	Página de dashboard	Figura 5.53 Figura 5.54
Static	style.css	Estilizar elementos escritos en HTML	

Tabla 5.10: Estructura de la interfaz de usuario.

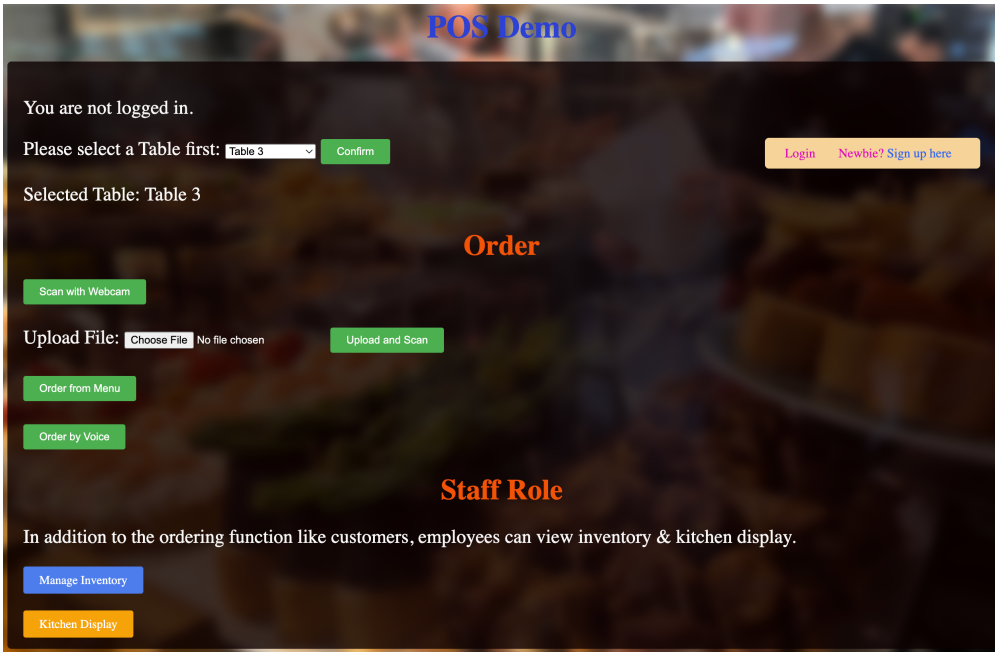


Figura 5.45: Página de inicio de POS de demostración.

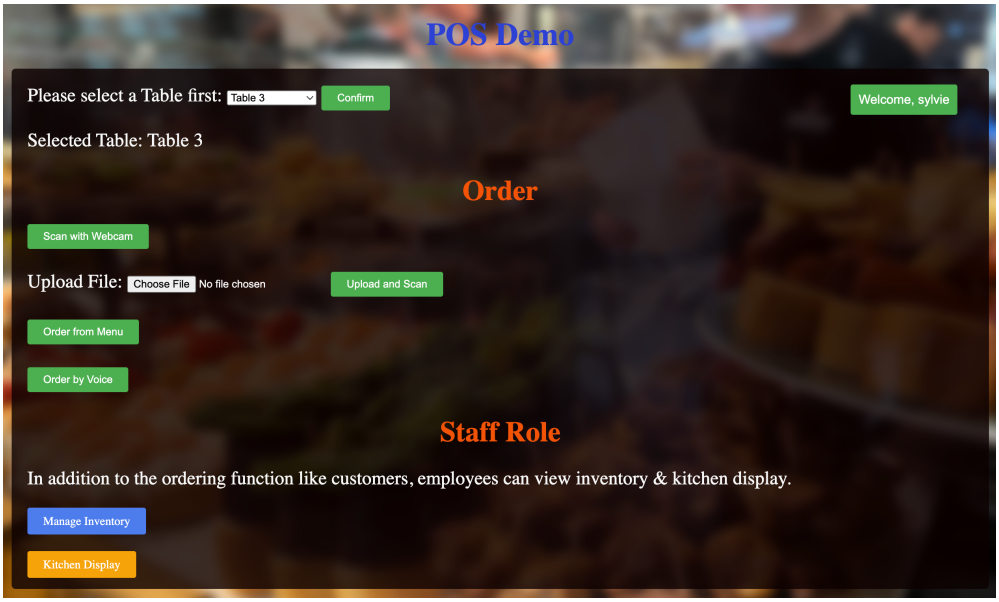


Figura 5.46: Página de inicio de login de POS de demostración.

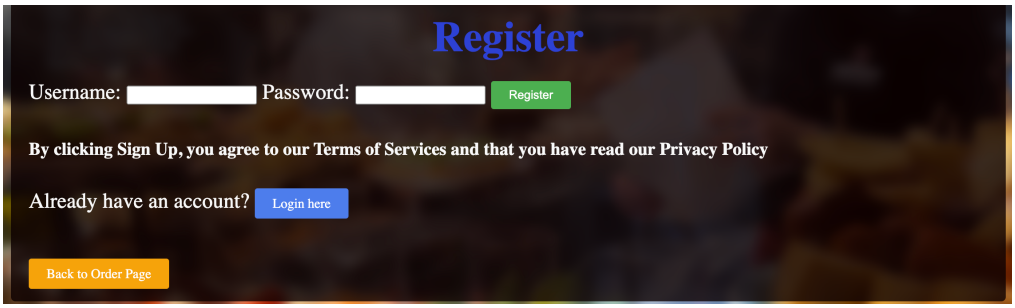


Figura 5.47: Página de registro.

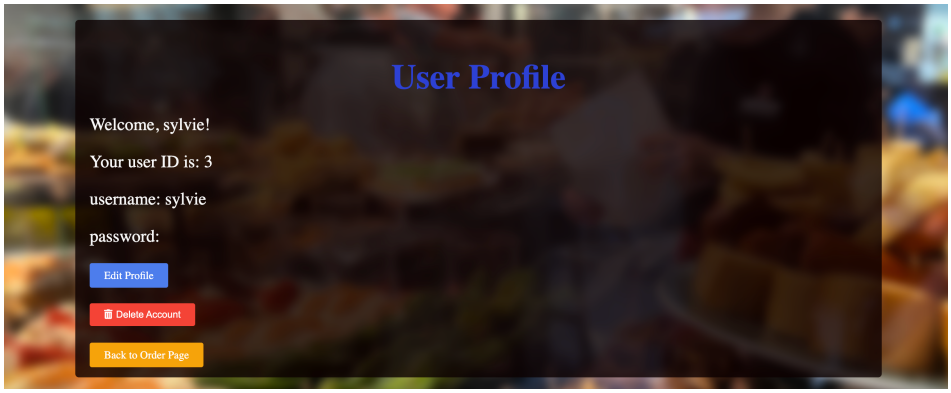


Figura 5.48: Página de perfil de usuario.

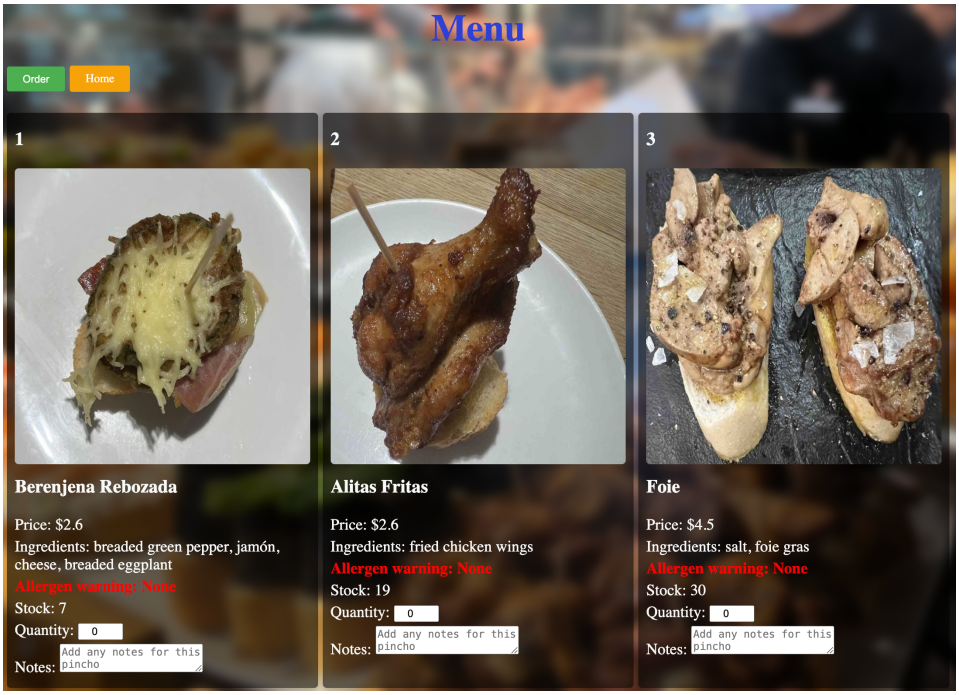


Figura 5.49: Página de menú.

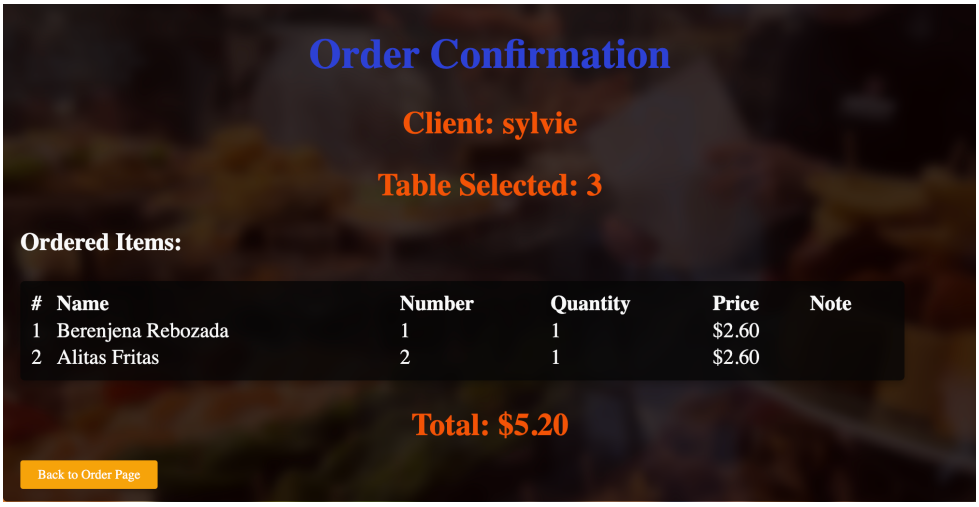


Figura 5.50: Página de confirmación del pedido.

Figura 5.51: Página de inventario.

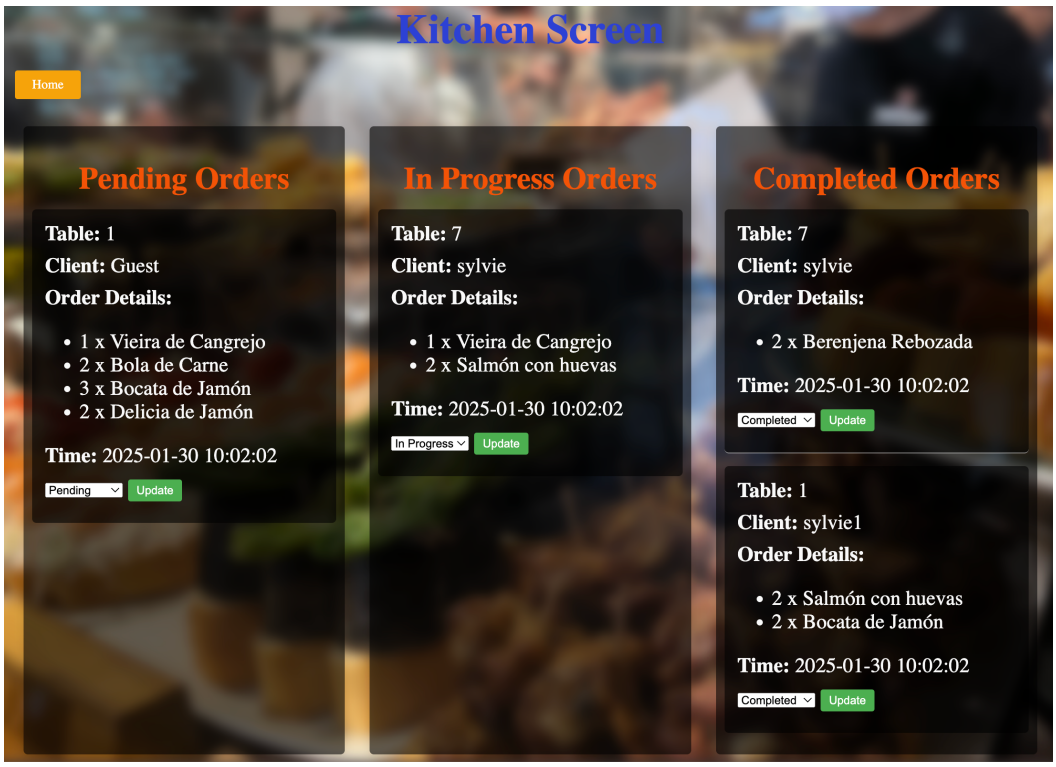


Figura 5.52: Página de la pantalla de cocina.

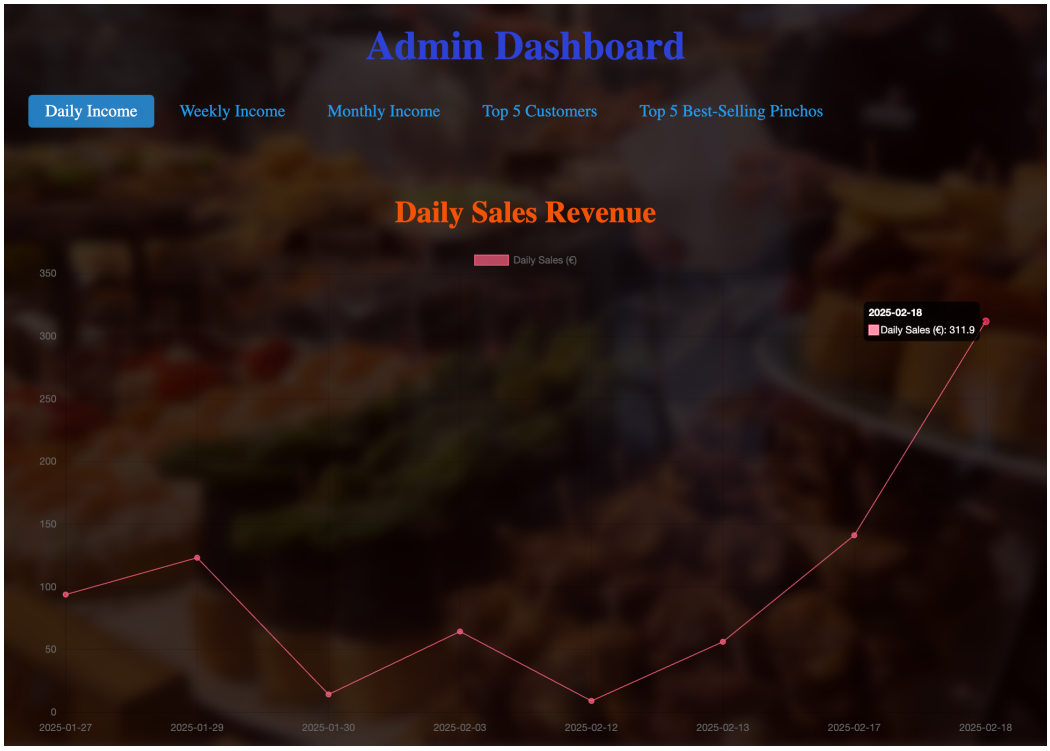


Figura 5.53: Página de Dashboard – Ingresos por ventas diarias.

Figura 5.54: Página de Dashboard
– Los 5 pinchos más vendidos a lo largo del tiempo.

Back-end (Flask)

El back-end maneja la lógica de la aplicación, las interacciones de la base de datos, la autenticación del usuario, el procesamiento de pedidos y la comunicación en tiempo real.

Rutas de Flask para manejar la autenticación de usuarios: registro, inicio de sesión, cierre de sesión.

Rutas para realizar pedidos:

- cargar: maneja la carga de archivos y la detección de elementos.
- escanear: capturar y procesar imágenes desde la cámara.
- voz: maneja entradas de voz para realizar pedidos.
- menú: orden desde la selección en el menú

Modelos de Base de Datos: Pintxo, TableOrder, Orders, User, OrderHistory y KitcheDisplay.

Detección de objetos: utilice el modelo YOLOv8 para detectar elementos en imágenes.

Actualizaciones en tiempo real: Flask-SocketIO para actualizaciones de pedidos en tiempo real.

Configuración

```
from flask import Flask, render_template, request, redirect, url_for, flash,
session, jsonify
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from sqlalchemy import text
from ultralytics import YOLO
import os
import cv2
import numpy as np
import speech_recognition as sr
import time  # Import time module
from datetime import datetime
import re  # Import re module for regular expressions
from models import db, Pintxo, TableOrder, Orders, User, OrderHistory,
KitchenDisplay
from flask_socketio import SocketIO, emit
from flask_migrate import Migrate
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity

app = Flask(__name__, static_url_path='/static')

# Set the path to the instance folder
instance_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
'instance')

# Database (pintxo.db)
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{os.path.join(instance_path,
"pintxos.db")}'

# Database (users.db)
app.config['SQLALCHEMY_BINDS'] = {
    'users': f'sqlite:///{os.path.join(instance_path, "users.db")}'
}

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.secret_key = '954f380e68f50589862a8e0d9cd475d3'

with app.app_context():
    db.create_all()

# Initialize SocketIO
socketio = SocketIO(app)
```

Esta configuración configura una aplicación web Flask con SQLAlchemy para la interacción de bases de datos, comunicación en tiempo real usando SocketIO e importa las bibliotecas necesarias para tareas como procesamiento de imágenes, reconocimiento de voz y cálculo de similitud de coseno. La aplicación se conecta a dos bases de datos: datos de pincho y datos de usuario.

3. Funcionalidad

3.1 Seguridad de los datos de autenticación del usuario

Se definen rutas para el registro de usuarios, inicio de sesión, edición de perfiles y eliminación de cuentas. Además, las contraseñas están codificadas por seguridad. Por lo tanto, los usuarios pueden ver su perfil, editar o eliminar su cuenta si ya no la utilizan.

Registro de usuario:

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
```

```
        password = request.form['password']
        hashed_password = generate_password_hash(password)
        new_user = User(username=username, password=hashed_password)
        try:
            db.session.add(new_user)
            db.session.commit()
            flash('User registered successfully!')
            return redirect(url_for('login'))
        except:
            flash('Username already exists!')
            return redirect(url_for('register'))
    return render_template('register.html')
```

Inicio y cierre de sesión de usuario:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if user and check_password_hash(user.password, password):
            session['user_id'] = user.id
            session['username'] = user.username  # Store username in session
            flash('Login successful!')
            return redirect(url_for('index'))
        else:
            flash('Invalid username or password!')
            return redirect(url_for('login'))
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    session.pop('username', None)  # Remove username from session
    flash('You have been logged out!')
    return redirect(url_for('index'))
```

Edición de perfil

```
@app.route('/edit_profile', methods=['GET', 'POST'])
def edit_profile():
    if 'user_id' not in session:
        flash('Please login to edit your profile!')
        return redirect(url_for('login'))

    user = User.query.get(session['user_id'])
```

```
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username:
            user.username = username
        if password:
            user.password = generate_password_hash(password)
        try:
            db.session.commit()
            flash('Profile updated successfully!')
            return redirect(url_for('profile'))
        except:
            db.session.rollback()
            flash('An error occurred while updating your profile.')
            return redirect(url_for('edit_profile'))

    return render_template('edit_profile.html', user=user)
```

Borrar la cuenta

Según la ley europea de protección de datos, el Reglamento General de Protección de Datos (GDPR), permite a los usuarios eliminar sus cuentas, lo que se conoce como "derecho de eliminación" o "derecho al olvido". Por tanto, los usuarios tienen derecho a solicitar la eliminación o eliminación de sus datos personales de la base de datos de la organización cuando ya no quieran utilizar el servicio.

Es por eso que cualquier aplicación tiene una opción que permite a los usuarios eliminar sus propias cuentas.

```
@app.route('/delete_account/<int:user_id>', methods=['POST'])
def delete_account(user_id):
    if 'user_id' in session and session['user_id'] == user_id:
        user = User.query.get(user_id)
        if user:
            db.session.delete(user)
            db.session.commit()
            flash('Your account has been deleted successfully.', 'success')
            session.clear()  # Clear session after deletion
            return redirect(url_for('index'))  # Redirect to index or login page
        else:
            flash('User not found.', 'error')
            return redirect(url_for('index'))
    else:
        flash('Unauthorized access or user ID mismatch.', 'error')
        return redirect(url_for('index'))
```

3.2 Integración YOLO

El modelo YOLO obtenido de las sesiones anteriores (5.2 Recopilación de datos y 5.3 Entrenamiento del modelo) se integra en la aplicación de demostración POS, lo que ayuda a detectar artículos (los pinchos) al cargar o escanear imágenes.

```
def load_yolo_model(model_path):
    model = YOLO(model_path)
    return model
model_path = "/Users/sylviedinh/Documents/Projects/ProjectTFM/POSDemo/best.pt"
model = load_yolo_model(model_path)
```

3.3 Proceso de realización de pedidos

Las rutas (@routes) se definen para manejar la colocación de pedidos, la visualización de menús y la carga de archivos para la detección de artículos basada en imágenes. Luego se muestran los artículos detectados y se calcula el costo total.

Los usuarios seleccionan una mesa y piden pinchos de diferentes maneras: suben una imagen, escanean la imagen del pincho con la cámara, ordenan desde el menú o por voz (dicen un número de pinchos).

```
# UPLOAD
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return redirect(request.url)

    file = request.files['file']
    if file.filename == '':
        return redirect(request.url)

    if file:
        filepath =
os.path.join('/Users/sylviedinh/Documents/Projects/ProjectTFM/POSDemo/uploads',
file.filename)
        file.save(filepath)

        detected_pintxos = detect_items(model, filepath)
        total = sum(item['price'] for item in detected_pintxos if item['stock'] >
0)
        if any(item['stock'] == 0 for item in detected_pintxos):
```

```
            flash("One or more items in your order are out of stock and cannot be
ordered.", 'error')
        return render_template('result.html', detected_pintxos=detected_pintxos,
total=total)

    return 'Error uploading file'

#SCAN
@app.route('/scan', methods=['GET', 'POST'])
def scan():
    if request.method == 'POST':
        try:
            cap = cv2.VideoCapture(0)
            if not cap.isOpened():
                return "Failed to open webcam."
            time.sleep(3)  # Wait for 3 seconds before capturing
            ret, frame = cap.read()
            if ret:
                image_path =
os.path.join('/Users/sylviedinh/Documents/Projects/ProjectTFM/POSDemo/uploads',
'scanned_image.jpg')
                cv2.imwrite(image_path, frame)
                cap.release()

                # Call function to detect items using YOLO model
                start_time = time.time()  # Measure time taken for detection
                detected_pintxos = detect_items(model, image_path)
                detection_time = time.time() - start_time  # Calculate detection
time
                total = sum(item['price'] for item in detected_pintxos if
item['stock'] > 0)

                # Check for out of stock items
                if any(item['stock'] == 0 for item in detected_pintxos):
                    flash("Out of stock and cannot be ordered.", 'error')

                flash(f"Image captured and processed in {detection_time:.2f}
seconds.", 'success')
                return render_template('result.html',
detected_pintxos=detected_pintxos, total=total)
            else:
                cap.release()
                return "Failed to capture image from webcam."
        except Exception as e:
            cap.release()
            return f"Error: {str(e)}"

    return render_template('scan.html')
```

Los dos tipos de órdenes son órdenes de escaneo y carga de imágenes, aprovechando la capacidad de reconocimiento de objetos del algoritmo YOLO, por lo que el sistema identificará esta imagen, se conectará a la base de datos (pintxos.db) con la información del pincho como el número, nombre, precio, ingredientes y stock.

Luego, el sistema verifica si algún artículo está agotado o no. Si hay stock, permitir a los clientes seleccionar y colocar ese artículo en el carrito..

```
#PLACE ORDER
@app.route('/place_order', methods=['POST'])
def place_order():
    table_number = session.get('selected_table')
    if not table_number:
        flash("Please select a table first.", 'error')
        return redirect(url_for('confirm_table'))

    username = session.get('username', 'Guest')

    orders = []
    total = 0

    # Iterate through the available pintxos to check the quantities
    for pintxo in Pintxo.query.all():
        quantity_key = f'quantity_{pintxo.number}'

        if quantity_key in request.form:
            quantity = int(request.form.get(quantity_key))

            if quantity > 0:
                if pintxo.stock >= quantity:
                    order = {
                        'number': pintxo.number,
                        'name': pintxo.name,
                        'ingredients': pintxo.ingredients,
                        'allergens': pintxo.allergens,
                        'price': pintxo.price,
                        'note': request.form.get(f'notes_{pintxo.number}', ''),
                        'image_url': pintxo.image_url,
                        'quantity': quantity
                    }
                    orders.append(order)
                    total += pintxo.price * quantity
                    pintxo.stock -= quantity
                else:
                    flash(f"Not enough stock for {pintxo.name}.", 'error')
                    return redirect(url_for('menu'))
    table = TableOrder.query.filter_by(number=table_number).first()
```

```
    if table:
        table.is_available = False
        db.session.commit()
    else:
        flash(f"Table {table_number} not found.", 'error')
        return redirect(url_for('menu'))

    # Save the active order to the KitchenDisplay table
    kitchen_display = KitchenDisplay(
        username=username,
        table_number=table_number,
        orders=orders,
        total=total
    )
    db.session.add(kitchen_display)

    # Save the order to history as well (but with 'pending' status)
    order_history = OrderHistory(
        username=username,
        table_number=table_number,
        orders=orders,
        total=total
    )
    db.session.add(order_history)

    db.session.commit()

    # Emit to the kitchen display
    socketio.emit('new_order', {'orders': orders}, namespace='/kitchen')

    return render_template('order_confirmation.html', orders=orders, total=total,
table_number=table_number, username=username)
```

3.4 Recomendador

Si no hay existencias, el sistema buscará artículos similares y recomendará platos de pincho con ingredientes similares a los clientes.

La función detect_items(model, image_path) utiliza el modelo YOLO para detectar pinchos en la imagen. Del resultado de la detección, extrae el nombre de clase previsto (número de pincho), consulta la base de datos de Pincho (pintxo) utilizando el número para encontrar la entrada de pincho correspondiente.

Comprueba el stock si encuentra algún pincho coincidente. Si el pincho está en stock (pintxo.stock > 0), crea una entrada en el diccionario (ID, número, nombre, precio, ingredientes, stock, etc.) para el pincho detectado.

Si el pincho está agotado (pintxo.stock == 0), busca pinchos similares basándose en la similitud de ingredientes usando find_similar_items_by_ingredients.

```
def detect_items(model, image_path):
    results = model(image_path)
    detected_items = {}

    for result in results:
        for box in result.boxes:
            item_number = result.names[int(box.cls)]
            pintxo = Pintxo.query.filter_by(number=item_number).first()

            if pintxo:
                item_id = pintxo.id
                if item_id not in detected_items:
                    if pintxo.stock > 0:
                        detected_items[item_id] = {
                            'id': pintxo.id,
                            'number': pintxo.number,
                            'name': pintxo.name,
                            'price': pintxo.price,
                            'ingredients': pintxo.ingredients,
                            'allergens': pintxo.allergens,
                            'stock': pintxo.stock,
                            'image_url': pintxo.image_url,
                            'notes': pintxo.notes,
                            'quantity': 1,
                            'recommendations': []}
                    else:
                        similar_items = find_similar_items_by_ingredients(pintxo)
                        recommendations = [item for item in similar_items if
item['id'] != pintxo.id]

                        detected_items[item_id] = {
                            'id': pintxo.id,
                            'number': pintxo.number,
                            'name': pintxo.name,
                            'price': pintxo.price,
                            'ingredients': pintxo.ingredients,
                            'allergens': pintxo.allergens,
                            'stock': 0,
                            'image_url': pintxo.image_url,
                            'notes': pintxo.notes,
                            'recommendations': recommendations}
    return list(detected_items.values())
```

La función find_similar_items_by_ingredients(pintxo) recorre en iteración todos los pinchos de la base de datos (Pintxo.query.all()) y excluye el pincho de entrada (item.id != pintxo.id).

Para cada pincho iterado, llama a compare_ingredients para calcular la puntuación de similitud de ingredientes entre el pincho actual y el iterado.

Si la puntuación de similitud es mayor que un umbral predefinido (0,5 en este caso), indica ingredientes similares.

```
def find_similar_items_by_ingredients(pintxo):
    similar_items = []
    all_items = Pintxo.query.all()

    for item in all_items:
        if item.id != pintxo.id:
            similarity_score = compare_ingredients(pintxo.ingredients,
item.ingredients)
            print(f"DEBUG: Comparing {pintxo.name} with {item.name}. Similarity
score: {similarity_score}")
            if similarity_score > 0.5:
                similar_items.append({
                    'id': item.id,
                    'number': item.number,
                    'name': item.name,
                    'price': item.price,
                    'ingredients': item.ingredients,
                    'allergens': pintxo.allergens,
                    'image_url': item.image_url,
                    'stock': item.stock,
                    'notes': item.notes,
                    'similarity_score': similarity_score
                })
    print(f"DEBUG: Found {len(similar_items)} similar items.")
    return sorted(similar_items, key=lambda x: x['similarity_score'],
reverse=True)
```

La función compare_ingredients(ingredientes1, ingredientes2) toma como entrada las cadenas de ingredientes de dos pinchos (ingredientes1 e ingredientes2).

Utiliza un CountVectorizer de la biblioteca scikit-learn para transformar las cadenas (strings) de ingredientes en vectores numéricos.

La función calcula la similitud del coseno entre los dos vectores de ingredientes usando cosine_similarity de scikit-learn. La similitud del coseno mide la similitud

entre dos vectores en función del ángulo entre ellos. Una puntuación más alta (más cercana a 1) indica composiciones de ingredientes más similares.

Devuelve la puntuación de similitud calculada entre las dos listas de ingredientes.

```
def compare_ingredients(ingredients1, ingredients2):
    # Tokenize ingredients into individual words or phrases
    vectorizer = CountVectorizer()
    ingredient_matrix = vectorizer.fit_transform([ingredients1, ingredients2])

    # Calculate cosine similarity between the ingredient vectors
    similarity_matrix = cosine_similarity(ingredient_matrix)
    similarity_score = similarity_matrix[0, 1]  # Get similarity score between
the two ingredient lists

    return similarity_score
```

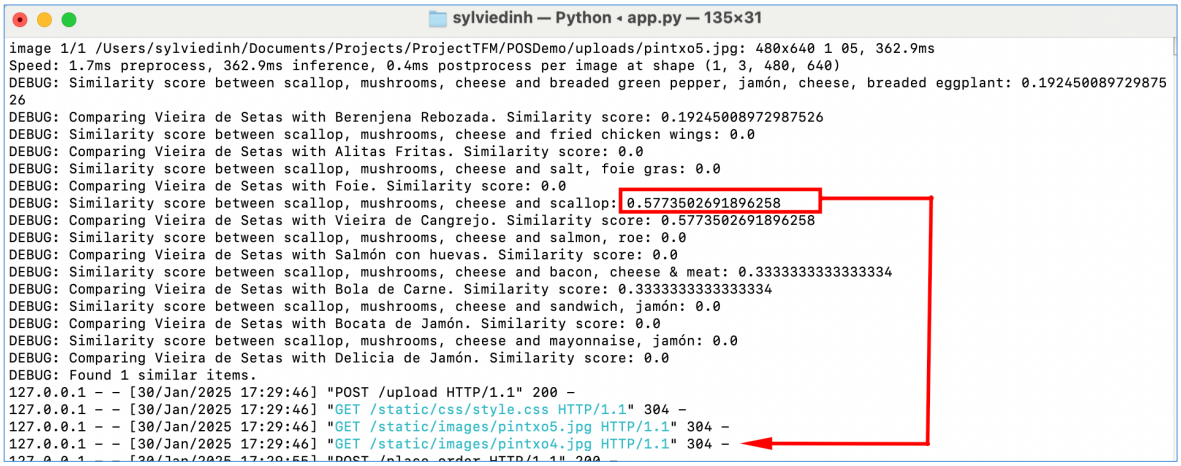


Figura 5.55: Puntuación de similitud entre los ingredientes de los pichos.

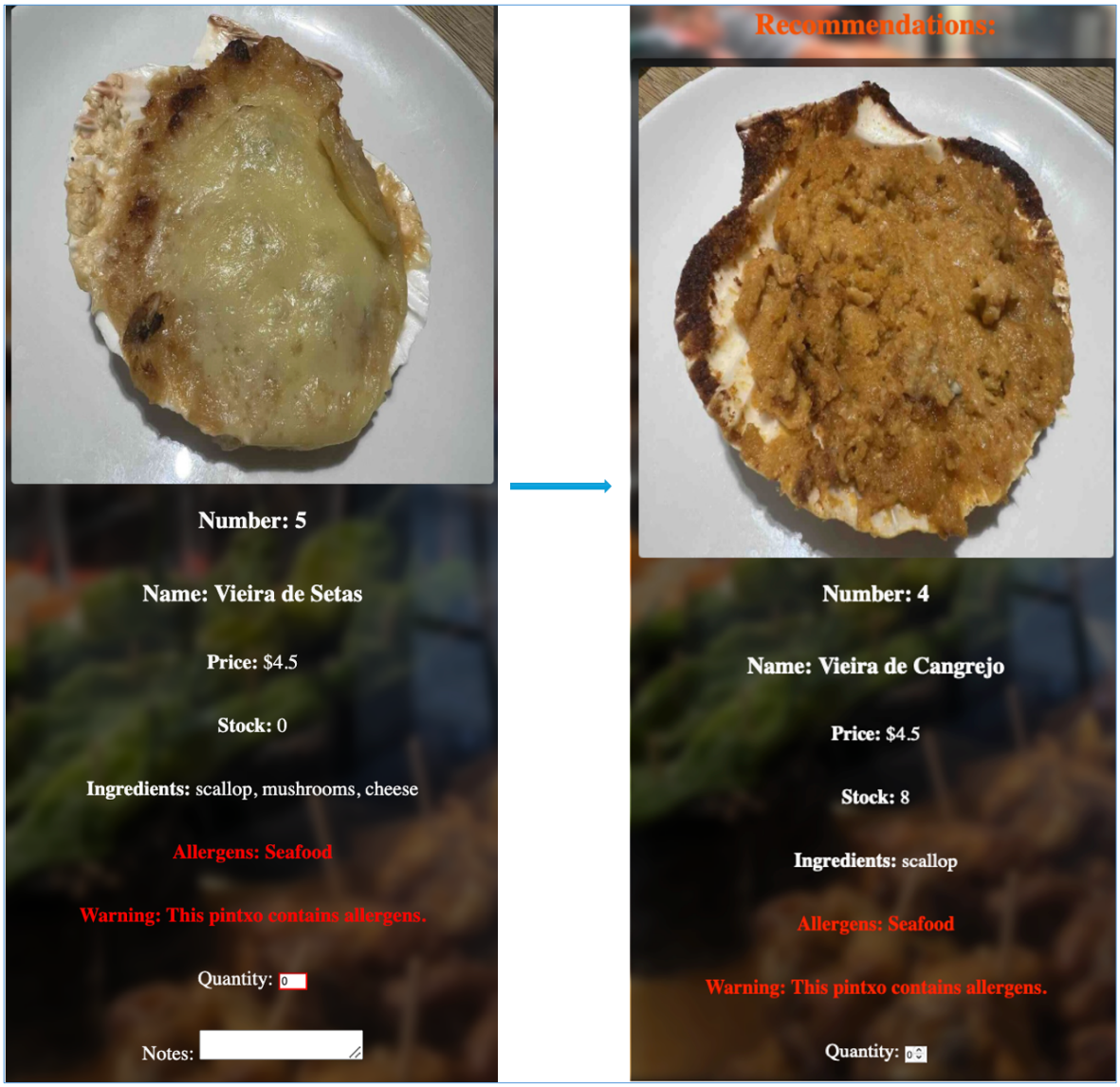


Figura 5.56: Resultado de una recomendación de artículo similar.

3.5 Funcionalidad adicional

Una ruta /inventario (/inventory route) permite gestionar el stock de pintxos (a modo demostrativo incluye una ruta de reseteo de stock).

```
#UPDATE INVENTORY
@app.route('/inventory')
def inventory():
    pintxos = Pintxo.query.all()
    for pintxo in pintxos:
        print(f"ID: {pintxo.id}, Name: {pintxo.name}, Stock: {pintxo.stock}")
    return render_template('inventory.html', pintxos=pintxos)
```

```
#KiTCHEN DISPLAY
@app.route('/kitchen')
def kitchen():
    # Fetch orders by status
    pending_orders = KitchenDisplay.query.filter(KitchenDisplay.status ==
'pending').all()
    in_progress_orders = KitchenDisplay.query.filter(KitchenDisplay.status ==
'in-progress').all()
    completed_orders = KitchenDisplay.query.filter(KitchenDisplay.status ==
'completed').all()

    # Pass the orders to the template
    return render_template('kitchen.html', pending_orders=pending_orders,
in_progress_orders=in_progress_orders, completed_orders=completed_orders)


#UPDATE ORDER STATUS
@app.route('/update_order_status/<int:order_id>', methods=['POST'])
def update_order_status(order_id):
    if 'username' not in session:  # Ensure the user is logged in
        flash('You need to log in first.', 'warning')
        return redirect(url_for('login'))

    # Fetch the order from KitchenDisplay by ID
    order = KitchenDisplay.query.filter_by(id=order_id).first()
    if order:

        new_status = request.form.get('status')
        if new_status:
            order.status = new_status
            db.session.commit()
            flash(f"Order status updated to '{new_status}'.", "success")
        else:
            flash("No status selected.", "error")
    else:
        flash("Order not found.", "error")

    return redirect(url_for('kitchen'))
```

```
#Reset the stock for testing purposes
from flask import jsonify
@app.route('/reset_stock', methods=['POST'])
def reset_stock():
    try:
        image_folder = 'static/images'
        initial_pintxos = [
            {"name": "Berenjena Rebozada", "number": "01", "price": 2.60,
"ingredients": "breaded green pepper, jamón, cheese, breaded eggplant",
"allergens":  "None", "stock": 10, "image_url": f"{image_folder}/pintxo1.jpg",
"notes": ""},
```

```
            {"name": "Alitas Fritas", "number": "02", "price": 2.60,
"ingredients": "fried chicken wings","allergens": "None", "stock": 20,
"image_url": f"{image_folder}/pintxo2.jpg", "notes": ""},
            {"name": "Foie", "number": "03", "price": 4.50, "ingredients": "salt,
foie gras", "allergens": "None", "stock": 30, "image_url":
f"{image_folder}/pintxo3.jpg", "notes": ""},
            {"name": "Vieira de Cangrejo", "number": "04", "price": 4.50,
"ingredients": "scallop", "allergens": "Seafood", "stock": 20, "image_url":
f"{image_folder}/pintxo4.jpg", "notes": ""},
            {"name": "Vieira de Setas", "number": "05", "price": 4.50,
"ingredients": "scallop, mushrooms, cheese", "allergens": "Seafood", "stock": 0,
"image_url": f"{image_folder}/pintxo5.jpg", "notes": ""},
            {"name": "Salmón con huevas", "number": "06", "price": 3.50,
"ingredients": "salmon, roe", "allergens": "Seafood", "stock": 10, "image_url":
f"{image_folder}/pintxo6.jpg", "notes": ""},
            {"name": "Bola de Carne", "number": "07", "price": 2.60,
"ingredients": "bacon, cheese & meat", "allergens": "Protein, Lactose", "stock":
20, "image_url": f"{image_folder}/pintxo7.jpg", "notes": ""},
            {"name": "Bocata de Jamón", "number": "08", "price": 4.50,
"ingredients": "sandwich, jamón", "allergens": "None", "stock": 30, "image_url":
f"{image_folder}/pintxo8.jpg", "notes": ""},
            {"name": "Delicia de Jamón", "number": "09", "price": 4.50,
"ingredients": "mayonnaise, jamón", "allergens": "None", "stock": 40,
"image_url": f"{image_folder}/pintxo9.jpg", "notes": ""}
        ]

        # Reset stock for each Pintxo to its initial value
        for pintxo_data in initial_pintxos:
            pintxo = Pintxo.query.filter_by(number=pintxo_data['number']).first()
            if pintxo:
                pintxo.stock = pintxo_data['stock']

        db.session.commit()
        return jsonify({'message': 'Stock reset successfully'}), 200

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500
```

5.5 Pruebas

La aplicación utiliza Flask-SocketIO para la comunicación en tiempo real entre el cliente (navegador web) y el servidor (aplicación Python).

Prueba local

- Comprobar que la aplicación Flask (app.py) esté configurada correctamente.
- Entorno virtual activo desde Terminal: source myenv/bin/activate
- Luego inicie el servidor de desarrollo de aplicaciones Flask usando el comando: "python3 app.py"
  (venv)~ % python3 ../ProjectTFM/POSDemo/app.py

Lanzará la aplicación en http://127.0.0.1:5000.

Acceso y prueba de puntos finales (testing Endpoints)

Navegamos a http://127.0.0.1:5000 en el navegador web para interactuar con la aplicación de demostración de POS.

Probamos varios puntos finales (rutas) ingresando las URL directamente en el navegador.

Debug
Depurar sentencias de impresión para rastrear el flujo de ejecución y comparar elementos:

```
print(f"DEBUG: Comparing {pintxo.name} with {item.name}. Similarity score:
{similarity_score}")

print(f"DEBUG: Found {len(similar_items)} similar items.")

print("DEBUG: No Pintxo found in database")

print("DEBUG: No valid number detected")
```

```
DEBUG: Voice input received: 'seven'
DEBUG: Detected number: 7
DEBUG: Found Pintxo: Bola de Carne
DEBUG: Similarity score between bacon, cheese & meat and breaded green pepper, jamón, cheese, breaded eggplant: 0.19245008972987526
DEBUG: Comparing Bola de Carne with Berenjena Rebozada. Similarity score: 0.19245008972987526
DEBUG: Similarity score between bacon, cheese & meat and fried chicken wings: 0.0
```

Figura 5.57: Declaraciones del registro de depuración (debug statements).

Instrucción:
Instrucción para utilizar la demostración de POS: Como usa POS Demo

Detección de objetos mediante vídeo con YOLO11:

YOLO11 es esencialmente una versión mejorada de YOLOv8, con una precisión media promedio (mAP) más alta en conjuntos de datos como COCO y utilizando un 22 % menos de parámetros que YOLOv8m. Por lo tanto, no se pueden omitir los mismos pasos de preparación de datos que para YOLOv8 (consulte la sección 5 de este documento). Al igual que el modelo YOLOv8, YOLOv11 utiliza imágenes etiquetadas (labelme2YOLO) para el entrenamiento y utiliza los resultados para reconocer imágenes/objetos en movimiento en videos, que en realidad son videos en directo/en tiempo real. (Aunque es posible utilizar los resultados del entrenamiento de YOLOv8 para reconocer objetos en videos, la diferencia es que YOLO11 es versión más nueva con un proceso de entrenamiento más rápido basado en su mejora).

Una pequeña demostración de detección de objetos en un video utilizando un modelo YOLOv11 que ha sido entrenado previamente en un conjunto de datos y se realizan predicciones para cada fotograma del video, según mi video aquí: video de reconocimiento de los pinchos utilizando YOLO11

Conclusiones y Líneas de trabajo futuras

Se ha construido un sistema POS (Punto de Venta) para la industria de alimentos y bebidas utilizando el modelo de detección de objetos YOLOv8. Se trata de una idea diferente en comparación con los sistemas POS convencionales. Este proyecto integró con éxito tecnología avanzada de visión por computadora en una aplicación del mundo real para mejorar la experiencia del cliente y la eficiencia operativa en la industria de alimentos y bebidas.

El sistema permite a los clientes cargar imágenes o escanear artículos usando la cámara de su propio dispositivo móvil para realizar un pedido. Esta característica no solo brinda la experiencia de "usar tecnología avanzada", sino que también hace que el proceso de pedir comida en el restaurante sea rápido y fácil, especialmente cuando el restaurante está lleno de gente. Además, ayuda a las empresas a reducir los costes laborales al reducir el personal de cajeros y camareros.

Este desarrollo del TFM no sólo está relacionado con el campo del procesamiento Big Data y la analítica de datos sino también con la inteligencia de negocio.

Si bien el conjunto de datos de 408 imágenes puede no ser lo suficientemente grande como para ser considerado "Big Data", los principios, técnicas y métodos aplicados en este proyecto son consistentes con las prácticas de Big Data: maneja la limpieza de datos (imágenes) corrigiendo imágenes con diferentes perspectivas. como rotación, desenfoque, cambio de tamaño y conversión en escala de grises; etiquetar las imágenes y convertirlas a YOLODataset para entrenar el modelo YOLOv8; entrenar el modelo YOLOv8 en 300 épocas puede escalar a conjuntos de datos más grandes.

Los datos del sistema POS se pueden incorporar a herramientas de inteligencia empresarial, analizando las operaciones comerciales y respaldando una toma de decisiones más precisa para desarrollar la estrategia y la visión empresarial.

La realización de este TFM implica una serie de objetivos personales, con un enfoque particular en aprender y dominar las diversas herramientas y tecnologías que juegan un papel importante en el desarrollo de aplicaciones: aprender a entrenar el modelo YOLOv8, procesar imágenes en Python, usar Labelme para etiquetar imágenes. y use labelme2YOLO para convertir anotaciones para garantizar que el conjunto de datos esté etiquetado y formateado para el modelo YOLO previo al entrenamiento, especialmente para aprender más sobre cómo desarrollar web con Flask.

Estos objetivos personales no sólo contribuyen a la finalización del TFM, sino que también me dotan de habilidades para futuros proyectos y carreras en los campos de la IA, el aprendizaje automático y la inteligencia empresarial.

Este proyecto puede abrir muchas vías para el desarrollo y la investigación futuros:

- Construir un sistema POS completo y avanzado como: automatizar completamente el sistema de pedidos de comida en el restaurante, desde que los clientes realizan pedidos en dispositivos móviles (propios o dispositivos en la mesa del restaurante), hasta eliminar por completo la impresión de recibos con una pantalla de cocina que actualiza el estado. en tiempo real. Además, los clientes pueden reordenar artículos de pedidos anteriores y comentar o votar por cada artículo directamente en la aplicación.
- Ampliación con otras funciones:
    - Reconocer la cara del cliente para iniciar sesión directamente en la aplicación. También se completaría el uso del reconocimiento facial para el pago.
    - Los clientes pueden realizar pedidos vía correo electrónico o servicio de mensajería (Lines, WhatsApp, …), el sistema recibirá automáticamente los pedidos y los enviará a la App del restaurante.

- Según el historial de pedidos del cliente (para usuarios registrados), se sugerirán nuevos platos similares a los pedidos habituales de los clientes.

- Si hay clientes que son alérgicos a algunos ingredientes, el sistema también admitirá sugerencias de platos similares sin estos ingredientes. Aplica para clientes registrados/iniciar sesión o no registrados/no iniciar sesión.

- Otra idea que se puede desarrollar a futuro de este proyecto es pedir a los clientes que califiquen el servicio en general y califiquen cada plato en particular.

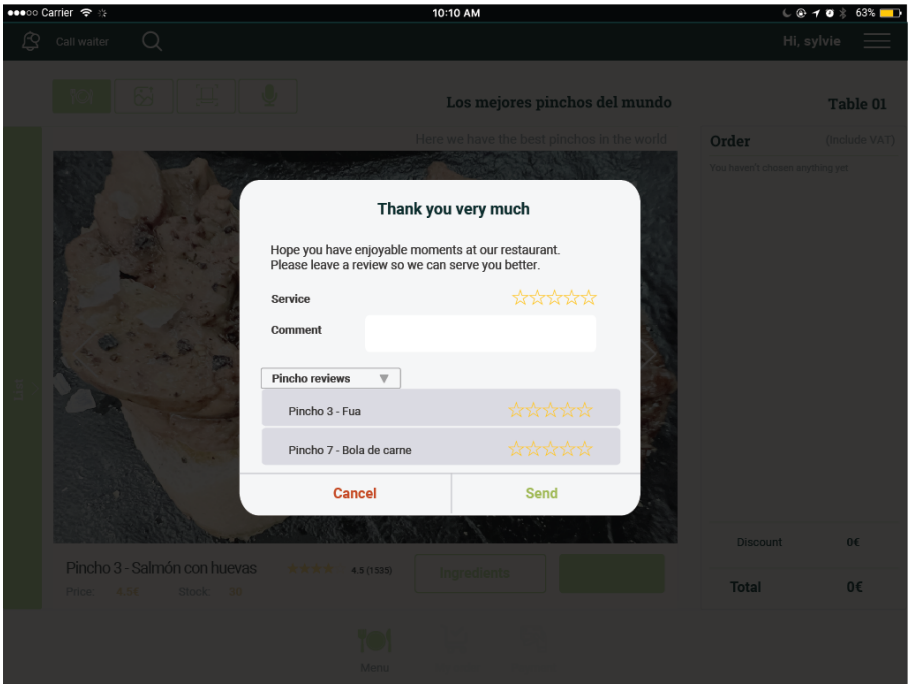


Figura 6.1: Revisión y votación del cliente.

- Integrar el robot al sistema POS, cuando esté entrenado, el robot puede reconocer imágenes de clientes, platos a pedir, sugerir platos similares e incluso servir para llevar platos a la mesa de los clientes. O de una manera más ambiciosa, los robots pueden hablar e interactuar directamente con los clientes, incluso entregarles comida.

- Integrar el sistema POS con dispositivos IoT (Internet of Things) como un sistema de inventario automático, utilizando únicamente el escaneo de imágenes en lugar de códigos de barras y RFID (Radio - frequency identification) como se hace hoy en día.

- Ampliar el sistema de aplicación POS desde pequeñas cafeterías hasta grandes cadenas de restaurantes y extenderlo al sector minorista donde se comercializan millones de artículos. El sector minorista se enfrenta a una formación sobre enormes cantidades de datos, lo que requiere más recursos y tecnología más avanzada.

- Soporte multilingüe en el uso del sistema POS para expandir el negocio a muchos países y regiones diferentes alrededor del mundo.

Apéndices

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Este apéndice refleja la planificación intermedia de las tareas a realizar durante el desarrollo del proyecto, así como el estudio de las consideraciones económicas y legales del mismo.

A.2. Planificación temporal

Este proyecto se ha desarrollado en el marco del TFM del Máster en Business Intelligence y Big Data en Entornos Seguros. Según su guía, el alcance y duración del TFM corresponde a 9 créditos ECTS, equivalentes a 225 horas.

El proyecto tiene una duración de varios meses desde principios de abril de 2024 hasta el 18 de febrero de 2025, realizándose diversas tareas durante este período. El detalle del trabajo realizado es el siguiente:

> **Fase 1**: Investigación. Aunque la idea de este proyecto surgió a principios de año con la lectura de muchos documentos, debido a limitaciones de tiempo, se investigó oficialmente a principios de junio. Las acciones de investigación incluyen diferentes posibilidades a la hora de llevar a cabo un proyecto en términos de tiempo, viabilidad o selección de herramientas y técnicas a utilizar para llevar a cabo con éxito las tareas a realizar a lo largo del proyecto..

Fase 2: Diseño. En esta fase se realizan el diseño, la implementación y la estructura del proyecto. Consta de 2 partes diseñadas diferentes: la primera parte es el flujo de trabajo de diseño y maquetas para demostrar la idea de cómo se verá el sistema POS en dispositivos móviles, eligiendo la pantalla del iPad para una visualización más fácil; La segunda parte se basa en el diseño del prototipo para crear una solución para construir un sistema POS de demostración en la aplicación web.

Fase 3: Implementación. Durante esta fase se llevaron a cabo diferentes partes del proyecto, como preparación de datos, limpieza de datos, entrenamiento del modelo YOLOv8 y finalmente desarrollo de aplicaciones web, una versión demo. El aprendizaje avanzado de herramientas y técnicas para el desarrollo de este proyecto también se lleva a cabo durante el diseño e implementación del proyecto. Esta aplicación tiene como objetivo demostrar que integrar el modelo YOLOv8 para el reconocimiento de objetos con Deep Learning en un sistema POS F&B es un proyecto factible y muy ambicioso.

Fase 4: Pruebas. Durante esta fase se realizaron pruebas para verificar el correcto funcionamiento de diversas actividades de desarrollo y establecer puntos de control para confirmar el correcto funcionamiento del sistema..

Fase 5: Documentación Durante esta fase se han documentado tanto informes, anexos como documentos relacionados con el desarrollo del proyecto.

A continuación se detallan los hitos de estas fases:

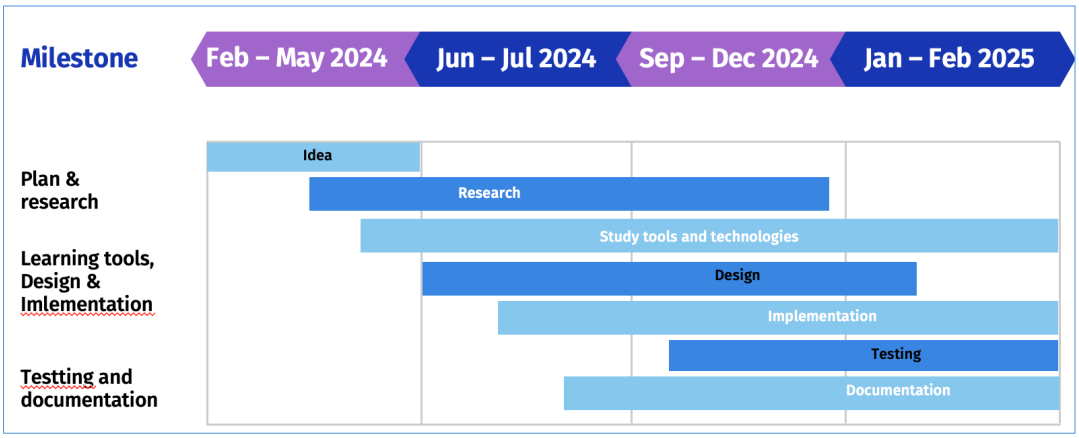


Figura A.1: TFM Milestone.

El método utilizado para gestionar el proyecto para TFM es Agile, un método de gestión de proyectos que implica dividir el proyecto en múltiples fases y enfatiza la colaboración y la mejora continua.

Trello es una herramienta que ayuda a construir, desglosar y gestionar las diferentes tareas involucradas en este proyecto.

A continuación, se muestra una captura de pantalla de la cuenta de Trello que estoy usando.

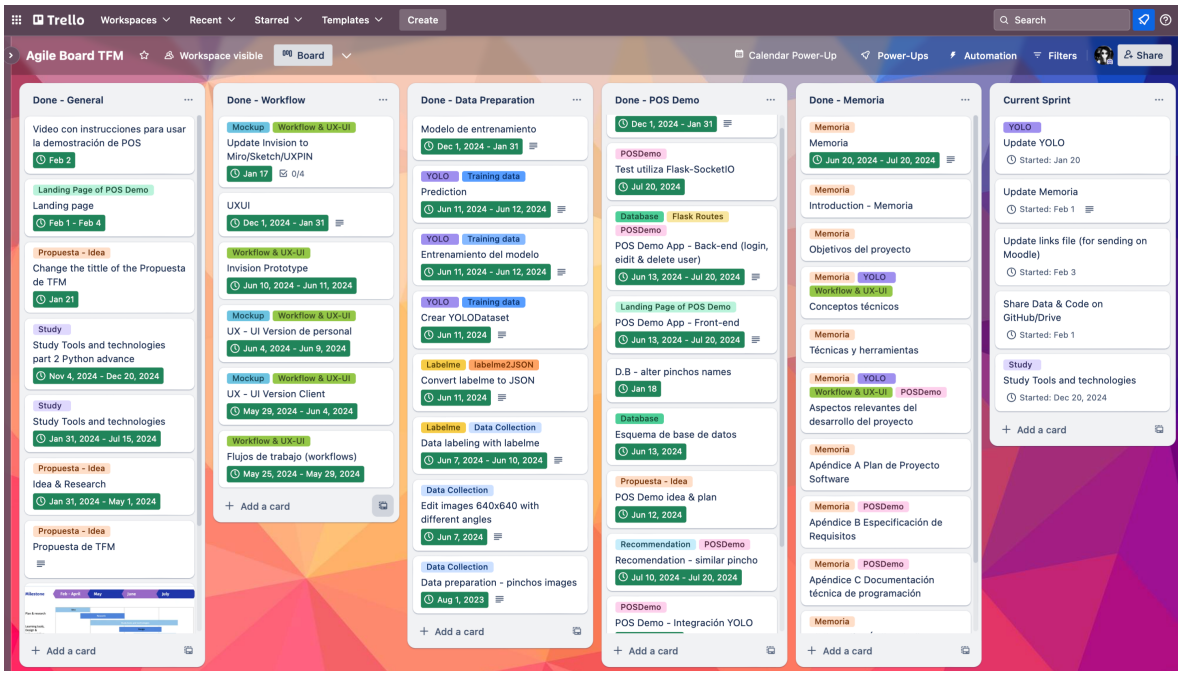


Figura A.2: Agile Board TFM.

A.3. Estudio de viabilidad

En esta sesión se detallará la viabilidad económica y legal de desarrollar este proyecto.

Estudio económico

En base al plan previo y los recursos técnicos utilizados podemos elaborar una estimación de la viabilidad económica del proyecto desarrollado. Para desarrollar el proyecto, tanto a nivel de documentación como de codificación, se realizó utilizando dos portátiles personales con las siguientes características:

Macbook Pro M3:

- Apple M3 Pro chip 11-core CPU, 14-core GPU, 16-core Neural Engine, 100GB/s memory bandwidth.
- 14.2-inch (diagonal) Liquid Retina XDR display.
- 32 GB unified memory, 1TB SSD.

Macbook Pro early 2015:

- 2.9GHz dual-core Intel Core i5 processor (Turbo Boost up to 3.3GHz) with 3MB shared L3 cache.
- Retina display 13.3-inch (diagonal), Intel Iris Graphics 6100.
- 8GB of 1866MHz LPDDR3 onboard memory.
- Storage 512GB PCIe-based flash storage.

Descripción	Mes	Costo mensual	Total
Recursos técnicos			
Ordenador personal Mac M3	8	29.20	233.60
Ordenador personal Mac Intel	8	12.50	100.00
Internet	8	42.00	336.00
Eléctrico	8	25.00	200.00
Recursos humanos			
Salario del analista programador	8	1,641.97	13,135.76
Contribución de la Seguridad Social	8	387.50	3100.00
Coste total			**16,235.76**

Tabla A.1: Presupuesto del proyecto.

Macbook Pro M3 es el equipo principal para desarrollar este proyecto. Macbook 2015 es para diseñar maquetas y flujos de trabajo debido a problemas de derechos de autor del software de diseño Adobe Illustrator.

Todos los costes presentados anteriormente (Tabla A.1) son únicamente estimaciones (unidad: €).

Respecto a los ordenadores personales, la ventaja de utilizar un Macbook es que el tiempo de uso suele ser mayor que otros ordenadores, al menos en mi opinión. Así, se estima que un portátil Macbook Pro tendrá un ciclo de vida de 10 años. El Macbook Intel cuesta 1500€, divididos en 10 años, por lo que la depreciación mensual es de aproximadamente 12,5. Mac M3 Pro es un modelo nuevo de Apple,

cuando lo compré a finales de noviembre del año pasado costaba 3505 €, la depreciación mensual se estima en 29,2 €.

El coste mensual a tiempo completo sería de 3.283,95 € y por tanto, en la modalidad a tiempo parcial, el coste correspondiente sería de 1.641,97 €. El coste total durante 8 meses será de 16.325 €. Finalmente habrá que sumar el coste de las cotizaciones a la seguridad social, en este caso según la base y tipo de cotización en 2024 será del 23,60%[1], el coste mensual será de 387,5€.

Viabilidad legal

En esta sección se discutirá la viabilidad legal del proyecto desarrollado como Trabajo Fin de Máster. Teniendo en cuenta los desarrollos que se han realizado, en la Tabla A.2 se detallan las licencias utilizadas.

Aplicación/Library	Licencia	Descripción
HTML5/ CSS	Licencia gratuita	
JavaScript	Licencia gratuita	
Flask/ SQLAlchemy	Licencia gratuita	Open source
Python	Licencia gratuita	Open source
Labelme	Licencia gratuita	Open source
Labelme2YOLO	Licencia gratuita	
YOLOv8, YOLOv11	Licencia gratuita	Ultralytics
Google Colab	Licencia limitada	Google cloud
Google Drive	Licencia limitada	Google
Adobe Illustrator	Licencia de pago	Adobe
InVision	Licencia de pago	InVisionApp Inc
UXPin	Licencia gratuita	UXPin
Microsoft Word	Licencia de pago	Microsoft student license
Visual Studio code	Licencia gratuita	Microsoft

Tabla A.2: Licencias utilizadas para desarrollar el proyecto.

[1] Bases y tipos de cotización 2024

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este apéndice presenta la especificación de requisitos para el desarrollo de un sistema POS de A&B en el sector de Alimentos y Bebidas, integrado con el modelo YOLOv8 para detección de objetos. El proceso de pedido de alimentos utiliza este sistema al permitir a los clientes cargar o escanear imágenes de alimentos para su detección y pedido automáticos.

B.2. Objetivos generales

- La tecnología de reconocimiento de objetos ayuda a automatizar el proceso de pedido de alimentos, reduciendo la entrada manual y reduciendo el tiempo de atención al cliente.
- Mejorar la experiencia del usuario con nuevas tecnologías.
- Sugerir platos similares para incrementar las ventas.
- Los datos recopilados se utilizan para análisis, elaboración de estrategias y toma de decisiones más precisas para las operaciones comerciales de la empresa.
- El sistema promete ampliarse a sucursales o a muchos campos diferentes.

B.3. Catalogo y especificación de requisitos

Las siguientes tablas muestran los requisitos del proyecto para una mejor comprensión por parte del lector.

Acciones	Estudio del caso de uso propuesto. Alcance y punto de partida.
Objetivos	Identificar el contexto manejado en el proyecto. Elegir herramientas para desarrollar nuestros objetivos.

Tabla B.1: Requisito 1: Evaluación inicial del proyecto.

Acciones	Instalar las herramientas necesarias. Configurar la configuración del entorno.
Objetivos	Activar y configurar las herramientas necesarias para el desarrollo de proyectos (LabelMe, labeme2YOLO, YOLOv8).

Tabla B.2: Requisito 2 - Operar el sistema de detección de objetos.

Acciones	Preparar y configurar Google Colab. Cargue conjuntos de datos de YOLO para entrenar el modelo en Google Drive.
Objetivos	Configure un entorno de trabajo para entrenar el modelo de detección de objetos.

Tabla B.3: Requisito 3: Inicialice el entorno de desarrollo en Google Colab conectándose al conjunto de datos en Google Drive.

Acciones	Entrenar el modelo YOLOv8 con conjuntos de datos etiquetados. Implementar el modelo en la aplicación de demostración POS.
Objetivos	Detecta y reconoce los pinchos en imágenes cargadas o escaneadas por los usuarios.

Tabla B.4: Funcionalidad 1: Detección y predicción de objetos utilizando best.pt del modelo de entrenamiento YOLOv8.

Acciones	Develop a system to process orders based on dish detection. Order confirmations.
Objetivos	Facilitate order processing from the POS application.

Tabla B.5: Funcionalidad 2: procesamiento de pedidos utilizando Flask (web framework).

Acciones	Automatizar la actualización del inventario en función de los pedidos realizados.
Objetivos	Mantenga un control preciso del inventario en tiempo real.

Tabla B.6: Funcionalidad 3 – Actualización de inventario.

Acciones	Desarrollar un sistema de recomendación de productos agotados. Sugiera pinchos similares según los ingredientes.
Objetivos	Proporcione soluciones alternativas a los usuarios cuando un producto no está disponible.

Tabla B.7: Funcionalidad 4 – Recomendador de artículos similares.

Acciones	Desarrollar la interfaz utilizando HTML5, CSS y JavaScript. Integre el frontend con Flask para conectar el frontend con el backend. Implemente la capacidad para que los usuarios carguen imágenes o escaneen elementos. Realizar pruebas de usabilidad para identificar y corregir problemas.
Objetivos	Provide an intuitive and efficient platform for order management. Facilitate user interaction with the object detection system. Ensure that all POS system functionalities are accessible and easy to use.

Tabla B.8: Funcionalidad 5: interfaz de usuario (una demostración de POS en una aplicación web que se ejecuta en el entorno Python).

Apéndice C

Documentación técnica de programación

C.1. Introducción

Este apéndice explicará los diversos pasos de desarrollo y componentes tomados para construir el sistema implementado en este proyecto: incluida la estructura de directorios, el manual del programador, los pasos de compilación del proyecto, la instalación, la ejecución y la prueba del sistema.

C.2. Estructura de directories

La estructura del directorio de desarrollo de este proyecto es la siguiente:

ProjectTFM: Este es el directorio raíz que contiene los directorios restantes además de los archivos responsables de configurar la implementación de todos los componentes, así como los archivos del servicio web. Contiene dos carpetas principales: Data and POSdemo (on local).

Data: Contiene imágenes para entrenar el modelo YOLOv8.

- Images: 51 imágenes originales.
- RetouchImages: Las imágenes fueron editadas desde diferentes ángulos.

- labelme_json_dir: Anotaciones para las imágenes.
- YOLODataset: conjunto de datos en formato YOLO generado a partir de las anotaciones.

La segunda parte son las carpetas y archivos necesarios cargados en Google Drive para entrenar el modelo YOLOv8, el directorio está organizado de la siguiente manera:

```
TFM/
|-- test/
|-- train/
|-- val/
|-- training YOLO8.ipynb
|-- dataset.yaml
|-- yolo8l.pt
|---
```

val, test, train y data.yaml on las carpetas y el archivo del conjunto de datos generados automáticamente a partir de la ejecución de labelme2YOLO (ejecutado con Python desde el escritorio) obtenidos de los pasos de preparación de datos anteriores (sesión 5.2), luego cargados en Google Drive.

yolo8l.pt es un archivo de peso previamente entrenado para el modelo YOLOv8, descargado de Ultralytics [6], preparado para el entrenamiento de reconocimiento de objetos y ejecutado en Google Colab. Este archivo contiene los parámetros aprendidos del modelo después de haber sido entrenado en un conjunto de datos grande (como COCO) y en nuestro conjunto de datos en este caso.

trainingYOLO8.ipynb: archivo de cuaderno de Jupyter utilizado para entrenar el modelo YOLOv8.

Una vez entrenado el modelo, YOLO crea automáticamente una carpeta 'ejecuciones' que se ponderará en la ruta 'run/train/weights' en Google Drive. El mejor peso se guardará como 'best.pt'; este modelo se descarga a la computadora personal para usarlo para la integración en la demostración de POS. Este archivo estará en el mismo directorio que app.py.

POSdemo: Este directorio contiene archivos y carpetas relacionados con la aplicación web, como CSS, modelo YOLO (best.pt) y base de datos utilizada en la aplicación.

- Carpetas:
  - static: Directorio para archivos estáticos como CSS e imágenes.
  - templates: directorio que contiene archivos de plantilla HTML (index.html, register.html, login.html, perfil.html, edit_profile.html, menu.html, upload.html, scan.html, voice.html, confirm_table.html,

result.html, order_confirmation.html, order_history.html, kitchen.html e inventory.html)
  - uploads: Carpeta para almacenar imágenes cargadas o escaneadas.
  - instance: Directorio que contiene la base de datos SQLite (pintxos.db y user.db).

- Ficheros:
  - best.pt: Modelo entrenado YOLOv8 utilizado para la detección de objetos.
  - models.py: Define modelos de bases de datos usando SQLAlchemy.
  - app.py: Punto de inicio (starting point) para ejecutar la aplicacion Flask.
  - requirement.txt: Lista de dependencias requeridas para el proyecto.

```
ProjectTFM/
|-- Data/
|      |-- Images/
|      |-- RetouchImages/
|      |-- labelme_jason_dir/
|      |      |-- YOLODataset/
|-- POSdemo/
|      |-- static/
|      |      |-- CSS/
|      |      |-- images
|      |-- templates/
|      |      |-- html files
|      |-- uploads/
|      |-- instance/
|      |      |-- pintxos.db
|      |      |-- user.db
|      |-- best.pt
|      |-- models.py
|      |-- app.py
|      |-- requirement.txt
|---
```

Los datos y código fuente del proyecto desarrollado se almacenan en el siguiente enlance: TFM

C.3. Compilación, instalación y ejecución del proyecto

1. Configure el entorno de desarrollo:

Para utilizar la aplicación, primero es necesario instalar todas las dependencias en equirement.txt:

```
Flask==2.1.1
Flask-SQLAlchemy==2.5.1
opencv-python==4.5.5.64
torch==1.11.0
```

```
ultralytics==8.0.3
SpeechRecognition==3.8.1
flask-socketio
```

- Crear un entorno virtual activo:

```
~ % source path/to/venv/bin/activate
```

- Luego instalar las dependencia:

```
pip install -r requirements.txt
```

o

```
pip install + 'library'
```

2. Inicialización de la base de datos:

```
db.init_app(app)

with app.app_context():
    db.create_all()
```

Inicializar la instancia de SQLAlchemy usando la aplicación Flask. Cree un contexto de aplicación para otorgar acceso a los recursos y la configuración específicos de la aplicación, y cree todas las tablas definidas en el modelo si aún no existen en la base de datos.

3. Ejecutar la aplicación Flask:

```
python3 app.py
```

Después de ejecutar este comando, solo tenemos que esperar a que accedamos al host local para interactuar con la aplicación web.

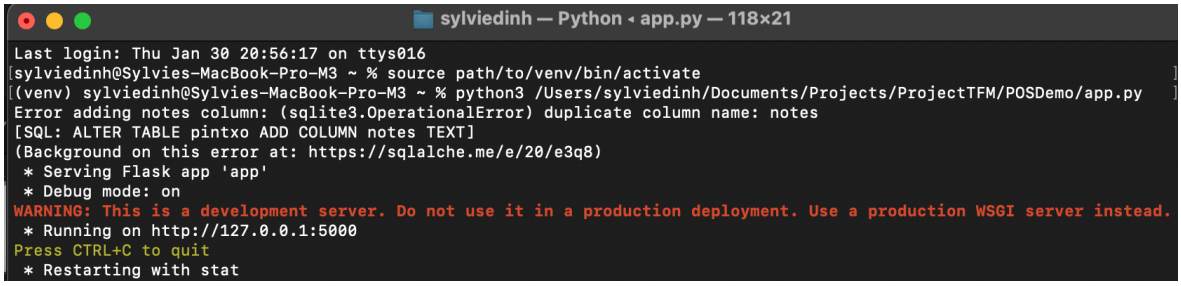


Figura C.1: Iniciación la aplicación de demostración.

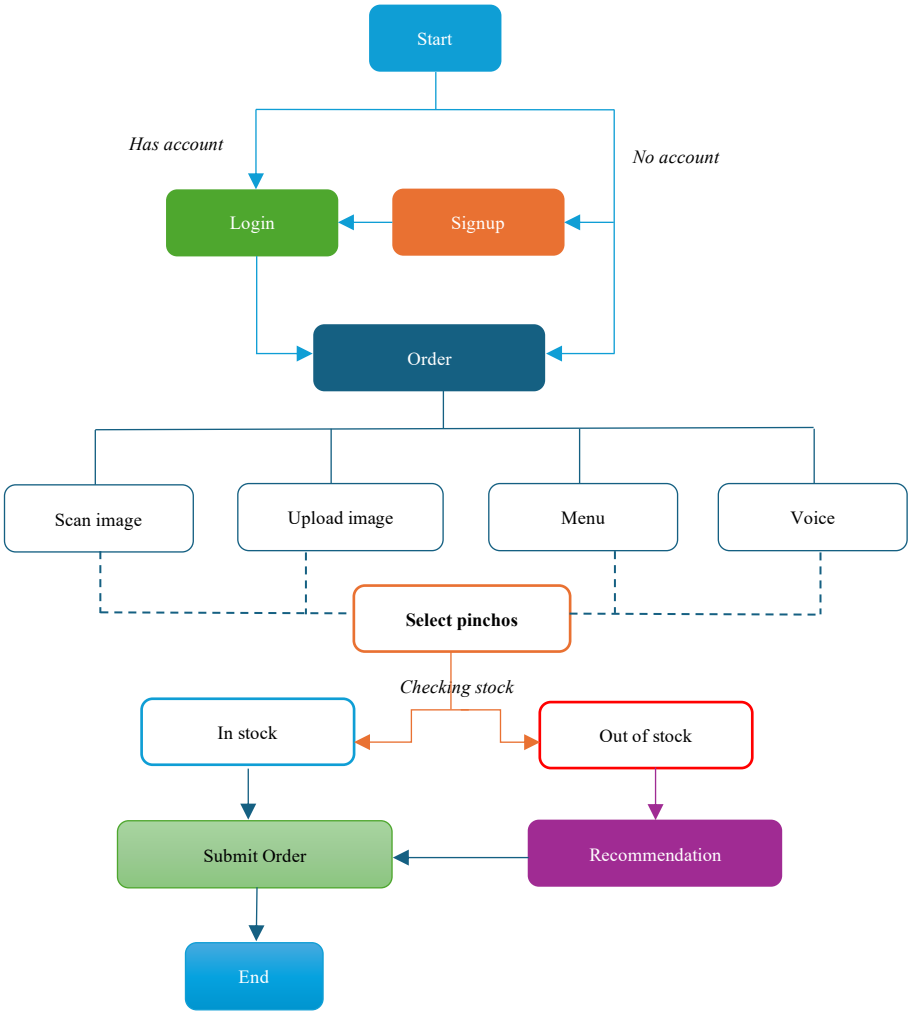
Luego, abra el navegador web y acceda a la dirección http://127.0.0.1:5000 para comenzar a utilizar el sistema POS de demostración para pedir comida.

** iniciar sesión por defecto: username: admin, password: admin*

La función de pedido se creó para procesar pedidos de muchas maneras diferentes, como el pedido tradicional desde el menú, el pedido por voz y, especialmente, probar escaneando o cargando imágenes de platos para que el sistema los reconozca automáticamente y recomendaciones de artículos similares.

El siguiente diagrama es el proceso de pedido del usuario en la demostración de POS. Además, los usuarios pueden ver su información personal, editar perfil y eliminar cuenta.

Proceso de pedido:



Bibliografía

[1] Wikipedia. *Flask (web framework)*, la enciclopedia libre, 2024. [Internet; accedido 20-junio-2024]. URL: https://en.wikipedia.org/wiki/Flask_(web_framework)

[2] Roboflow. *Explore Ultralytics YOLOv8*, 2023 [Internet, accedido 06-junio-2024]. URL: https://yolov8.com

[3] Ultralytics. *Ultralytics YOLO Docs*, 2023. [Internet, accedido 30-mayo-2024]. URL: https://docs.ultralytics.com

[4] Data Science. You Only Look Once (YOLO): What is it?, 2024. [Internet, accedido 20-mayo-2024]. URL: https://datascientest.com/en/you-only-look-once-yolo-what-is-it#:~:text=You%20Only%20Look%20Once%20or,the%20mainstays%20of%20computer%20vision.

[5] Ultralytics. *Ultralytics YOLO Docs – Benchmark*, 2023. [Internet, accedido 20-mayo-2024]. URL: https://docs.ultralytics.com/usage/python/#benchmark

[6] Ultralytics. *Ultralytics YOLO Docs*, 2024. [Internet (Github), accedido 20-mayo-2024] URL: https://github.com/ultralytics/ultralytics

[7] Ultralytics. *Ultralytics YOLO Docs - Command Line Interface Usage*, 2023. [Internet, accedido 20-mayo-2024]. URL: https://docs.ultralytics.com/usage/cli/#val

[8] Wkentaro. *Image Polygonal Annotation with Python*, 2018. [Internet (Github), accedido 02-junio-2024]. URL: https://github.com/labelmeai/labelme

[9] OVision. *A quick but comprehensive guide to LabelMe - an image/video annotation tool for deep learning*, 2022. [Internet (Youtube), accedido 02-junio-2024]. URL: https://www.youtube.com/watch?v=HiW3qeJVQCg

[10] Wkentaro. *Labelme*, 2018. [Internet (Github), accedido 09-junio-2024]. URL: https://github.com/wkentaro/labelme/releases

[11] Roboflow – Joseph Nelson. How to Label Image Data for Computer Vision Models, 2024. [Internet, accedido 02-junio-2024]. URL: https://blog.roboflow.com/tips-for-how-to-label-images/

[12] Roboflow – James Gallagher. How to Use LabelMe: A Complete Guide, 2023. [Internet, accedido 02-junio-2024]. URL: https://blog.roboflow.com/labelme/#:~:text=LabelMe%20is%20an%20open%20source,circle%2C%20line%2C%20and%20points

[13] Python Software Foundation. *Python Package Index, Labelme2YOLO*, 2024. [Internet, accedido 09-junio-2024]. URL: https://pypi.org/project/labelme2yolo/

[14] Roboflow – Piotr Skalski. *How to Train YOLOv8 Object Detection on a Custom Dataset*, 2023. [Internet, accedido 11-junio-2024]. URL: https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/

[15] Roboflow - Jacob Solawetz & Francesco. *What is YOLOv8? The Ultimate Guide. [2024] What's new in YOLOv8*, 2023. [Internet, accedido 20-junio-2024]. URL: https://blog.roboflow.com/whats-new-in-yolov8/

[16] Medium, ODSC - Open Data Science. *Overview of the YOLO Object Detection Algorithm*, 2018. [Internet, accedido 02-julio-2024]. URL: https://odsc.medium.com/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0

[17] Open MMLab. *Docs – Algorithm principles and implementation with Yolov8.* [Internet, accedido 02-julio-2024]. URL: https://mmyolo.readthedocs.io/en/latest/recommended_topics/algorithm_descriptions/yolov8_description.html#:~:text=In%20summary%2C%20YOLOv8%20is%20a,achieve%20new%20levels%20of%20performance.

[18] Github – RangeKing. *Brief summary of YOLOv8 model structure, 2023.* [Internet, accedido 02-julio-2024]. URL: https://github.com/ultralytics/ultralytics/issues/189

[19] Towards Data Science - Pratheesh Shivaprasad. *A Comprehensive Guide To Object Detection Using YOLO Framework — Part I*, 2018. [Internet, accedido 11-junio-2024]. URL: https://towardsdatascience.com/object-detection-part1-4dbe5147ad0a

[20] Medium – Paul Lefevre, *Part 1: ID Documents Detection with YOLOv8 and Orientation Correction, 2023*. [Internet, accedido 11-junio-2024]. URL: https://medium.com/@paul_lefevre/id-documents-detection-with-yolov8-plus-rotation-e991192e74d2

[21] Medium - Siddheshwar Harkal. *Image Classification with YOLOv8*, 2023. [Internet, accedido 11-junio-2024]. URL: https://sidharkal.medium.com/image-classification-with-yolov8-40a14fe8e4bc

[22] Mindy McAdams Revision. *Docs>>Flask Templates*. [Internet, accedido 20-junio-2024].URL: https://python-adv-web-apps.readthedocs.io/en/latest/flask3.html#:~:text=Folder%20structure%20for%20a%20Flask%20app&text=Everything%20the%20app%20needs%20is,%2C%20JavaScript%20files%2C%20and%20images.

[23] Faun - Manibhadra Singh Rathore. *From Zero to Hero: Creating a Food Delivery App with Python*, 2023. [Internet, accedido 14-junio-2024]. URL: https://faun.dev/c/stories/manibhadra/from-zero-to-hero-creating-a-food-delivery-app-with-python/

[24] Flask. User's guide. [Internet, accedido 15-junio-2024]. URL: https://flask.palletsprojects.com/en/3.0.x/

[25] Real Python – Phillipp Acsany. *Build a Scalable Flask Web Project From Scratch*, 2023. [Internet, accedido 15-junio-2024]. URL: https://realpython.com/flask-project/

[26] GeeksforGeeks. Flask Tutorial, 2024. [Internet, accedido 15-junio-2024]. URL: https://www.geeksforgeeks.org/flask-tutorial/

[27] InVision. *Buiding Prototypes – Intro to building prototypes*, 2023. [Internet, accedido 26-junio-2024]. URL: https://support.invisionapp.com/docs/intro-to-building-prototypes

[28] Adobe. *Illustrator User Guide*, 2024. [Internet, accedido 22-junio-2024]. URL: https://helpx.adobe.com/illustrator/user-guide.html

[29] Jiawei Han, Micheline Kamber and Jian Pei. *Data Mining: Concepts and Techniques - A volume in The Morgan Kaufmann Series in Data Management Systems*. Chapter 2 - Getting to know your data, 2012.

[30] UXPin. *What is Prototype – A guide to functional UX*, 2024. [Internet, accedido 10-diciembre-2024]. URL: https://www.uxpin.com/studio/blog/what-is-a-prototype-a-guide-to-functional-ux/

[31] Ultralytics. *Ultralytics YOLO 11,* 2024. [Internet, accedido 20-diciembre-2024]. URL: https://docs.ultralytics.com/models/yolo11/#what-tasks-can-yolo11-models-perform

[32] Roboflow. *What is YOLO 11*, 2024. Internet, accedido 30-enero-2025]. URL: https://blog.roboflow.com/what-is-yolo11/