Universidad de Castilla-La Mancha
CAMPUS DE EXCELENCIA INTERNACIONAL

Facultad de Ciencias Sociales
Talavera de la Reina. UCLM
CAMPUS DE EXCELENCIA INTERNACIONAL

Gestión del Proyecto

NextGen Software

Sara Gutiérrez Caja

Ángela Jiao Peng

Ingrid Niveiro Ben

Houda El Ouahabi Khirat

Índice

1. Introducción

Este proyecto desarrolla un sistema para la gestión de pedidos y repartos de comida a domicilio, que tiene como objetivo facilitar y optimizar la entrega de pedidos a domicilio. Los usuarios pueden registrarse como Cliente, Restaurante o Repartidor, teniendo cada uno funcionalidades específicas.

2. Características Principales

- **Usuarios anónimos**: Registro en el sistema y búsqueda de restaurantes sin autenticación.
- **Restaurante**: Gestión del menú y actualización de precios.
- **Cliente**: Realización de pedidos, selección de restaurantes, favoritos y gestión de la entrega.
- **Repartidor**: Recepción de notificaciones para la recogida y entrega de pedidos, registro de cada etapa.

3. Requisitos funcionales/No funcionales

3.1 Restaurante

3.3.2 Requisitos funcionales

- **Poder dar de alta el menú:** El restaurante debe de crear los ítems para poder introducirlos en un menú.
- **Modificar menú:** Introducir ítems y eliminarlos.

3.2 Cliente

3.2.1 Requisitos funcionales

- **Buscar restaurante**: El cliente puede marcar un restaurante como favorito y se añade a la lista de favoritos. Por otra parte, el cliente puede eliminar el restaurante de su lista de favoritos.
- **Seleccionar restaurante:** El cliente puede seleccionar un restaurante para visualizar su menú y sus productos.
- **Seleccionar menú:** Cuando el cliente lo desee, puede seleccionar diversos productos de un menú de un restaurante para poder llevar a cabo un pedido.
- **Realizar pedidos:** El cliente puede realizar un pedido de los productos que desee de un restaurante determinado introduciendo sus datos correspondientes.

3.2.2 Requisitos no funcionales

- **Seleccionar repartidor:** El sistema debe de selección un repartidor determinado según los datos introducidos por el cliente.
- **Notificar al repartidor:** El sistema debe de notificar al repartidor de que tiene que llevar a cabo una nueva entrega.

3.3 Repartidor

3.3.1 Requisitos funcionales

- **Registrar recogida del pedido:** El repartidor debe de registrar si ha recogido un pedido en su determinado restaurante.
- **Registrar entrega del pedido:** Tras realizar la entrega, el repartidor debe de notificar al sistema de que se ha llevado a cabo con éxito el pedido.

3.4 Usuario anónimo

3.4.1 Requisitos funcionales

- **Darse de alta como usuario registrado:** El usuario puede registrarse con un rol de cliente, repartidor o como un restaurante.
- **Autentificarse:** El usuario puede acceder a su vista correspondiente con sus funcionalidades correspondiente introduciendo sus datos y contraseña.

5. Planificación

5.1 Metodología Utilizada

Para la gestión del desarrollo de nuestro proyecto, hemos adoptado la metodología ágil 'Scrum', organizando nuestro trabajo en **sprints de una/dos semanas**.

La metodología 'Scrum' nos aporta las siguientes ventajas:

- **Iteraciones Cortas/Medianas:** Realización de entregas continuas de funcionalidades, lo que facilita la adaptación a cambios y permite obtener feedback rápido entre todos los integrantes.
- **Planificación Efectiva:** Antes de cada sprint, realizamos una reunión de planificación donde definimos los objetivos y las tareas a completar, en el caso de que un integrante no lleve a cabo las tareas que le corresponde, se le asigna a otra persona.
- **Visibilidad del Progreso:** Al final de cada sprint, se presentan los resultados, lo que aumenta la transparencia y permite al equipo recibir feedback inmediato sobre las funcionalidades implementadas.

5.2 Cronología

En nuestro caso, hemos utilizado GitHub Projects como una herramienta central para organizar y gestionar nuestras tareas a lo largo del ciclo de desarrollo del proyecto. GitHub Project nos ha permitido implementar un flujo de trabajo en un tablero Kanban en el que se divide las tareas en diferentes columnas según su estado actual. En nuestro caso tenemos varias columnas.

- **Backlog:** En esta columna se describen las tareas a realizar, pero no están asignadas para ser trabajadas.
- **Product-Backlog**: En este apartado se posicionan las tareas ya asignadas con su correspondiente prioridad para comenzar a ser trabajadas.
- **In progress:** Muestra las tareas que se está trabajando activamente.
- **In Review:** Se posicionan las tareas que han sido completadas, pero están pendiente de revisión por parte de todos los integrantes.
- **Done:** Contiene las tareas finalizadas y fusionadas en su rama correspondiente.

En este apartado, se incluyen capturas de las tareas posicionadas en las columnas Backlog y **Product-Backlog** al inicio de cada sprint.

- **Sprint 1:**

  En este sprint, las tareas estaban repartidas entre tres personas debido a que una de las contribuyentes actuales no se encontraba en nuestra empresa.

  - **Fechas:** 07/10/2024 - 14/10/2024

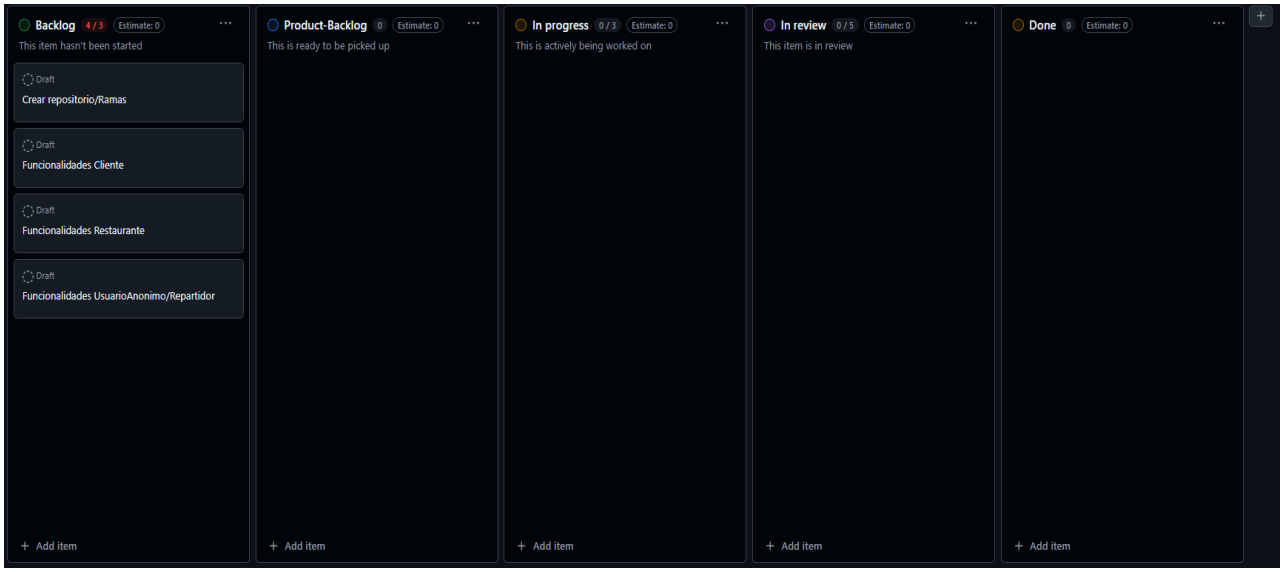


Ilustración 1: Backlog Sprint 1

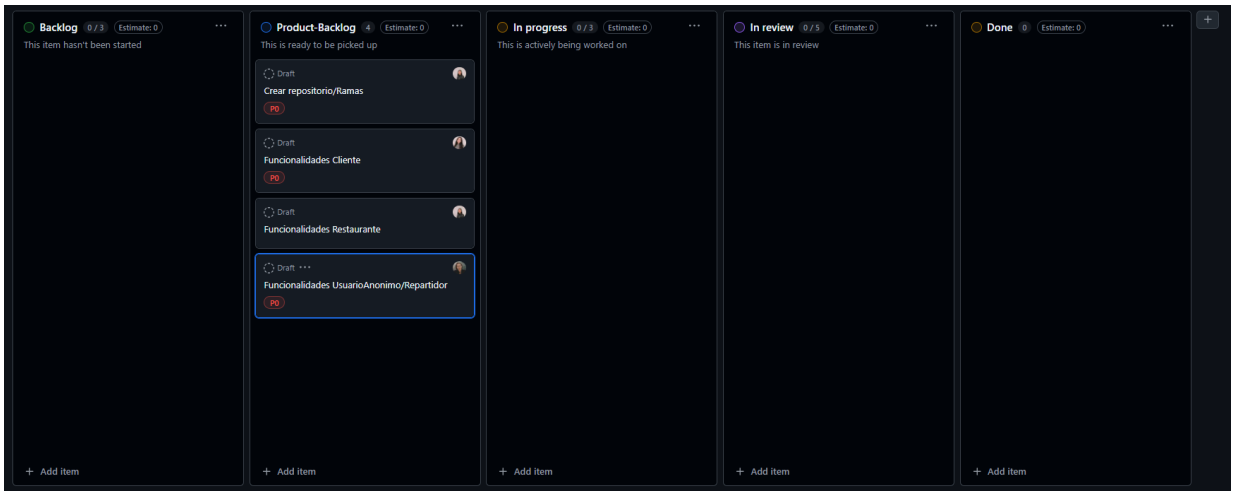


Ilustración 2: Product Backlog sprint 1

Finalmente, todas las tareas se incorporarán en la columna 'Done'.

- **Sprint 2:**
  - **Fechas:** 14/10/2024 - 21/10/2024

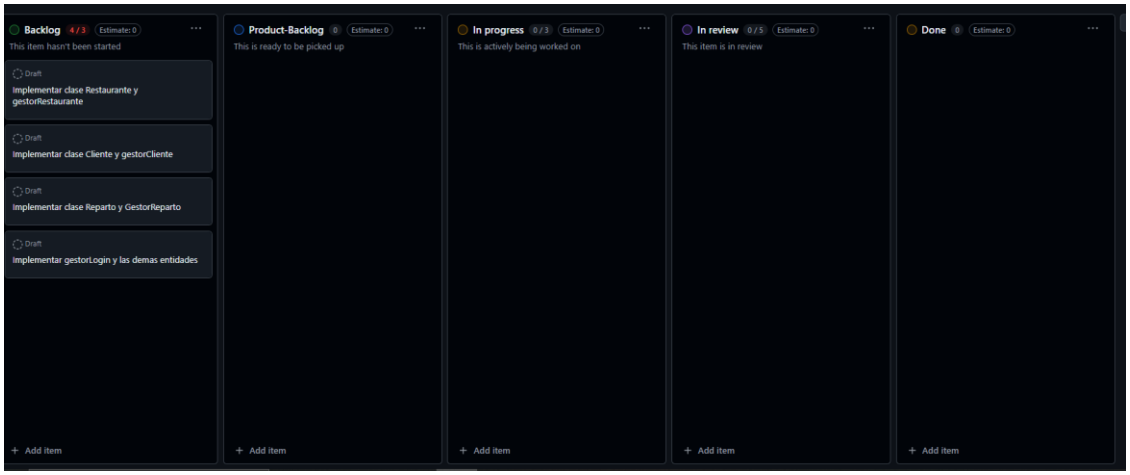


Ilustración 3: Backlog Sprint 2

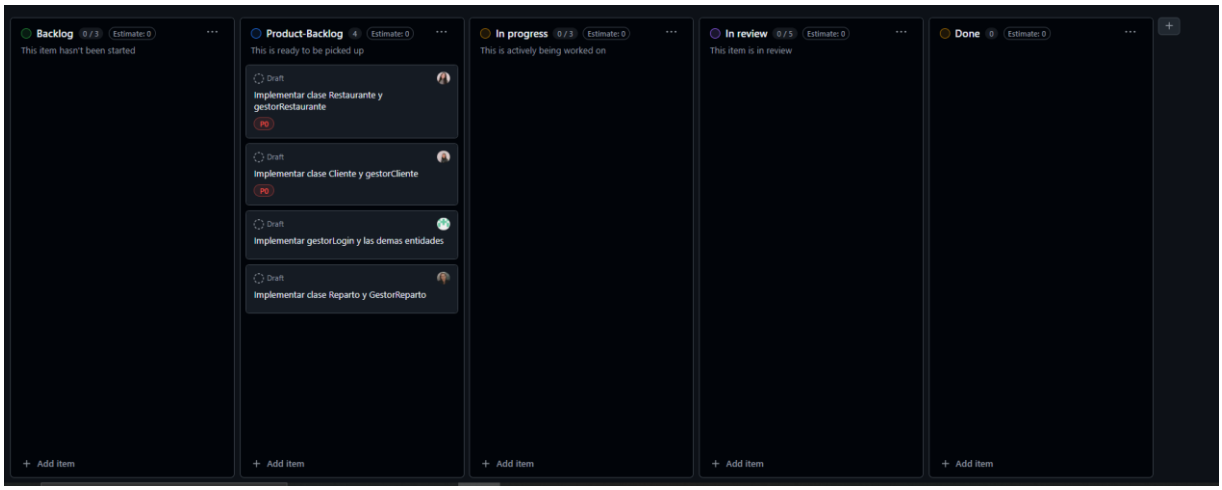


Ilustración 4: Product Backlog Sprint 2

- **Sprint 3:**
  - **Fechas:** 22/10/2024 - 5/11/2024

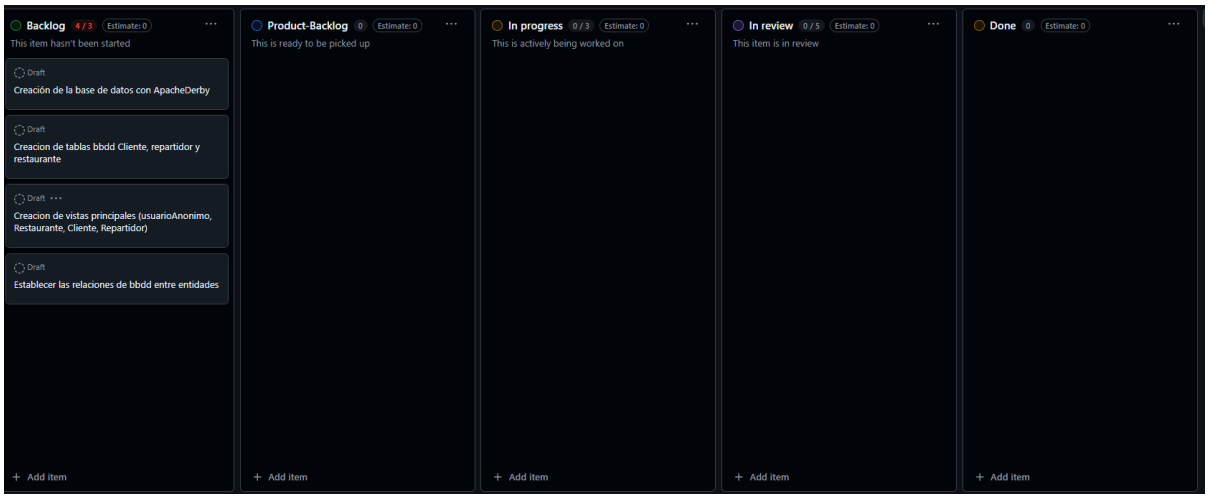


Ilustración 5: Backlog Sprint 3

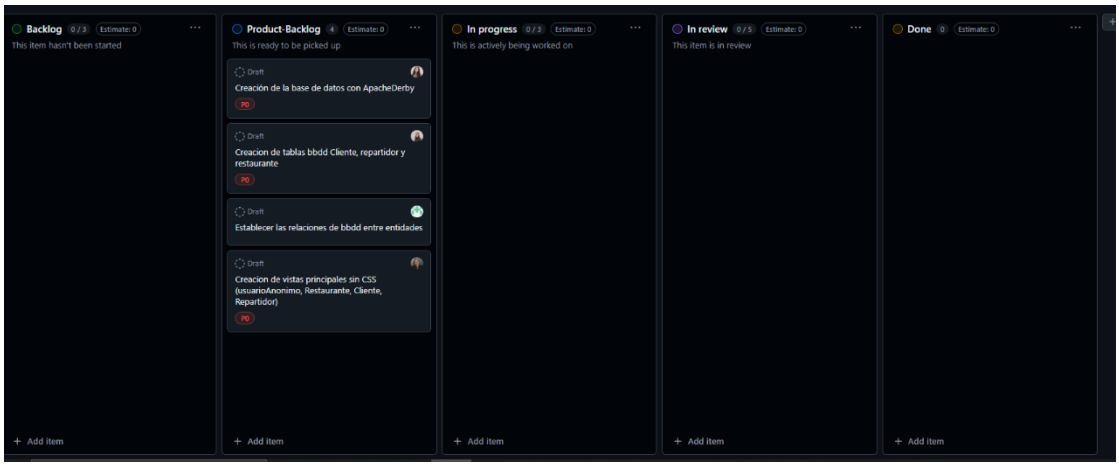


Ilustración 6: Product Backlog Sprint 3

- **Sprint 4:**
  - **Fechas:** 20/11/2024-27/11/2024

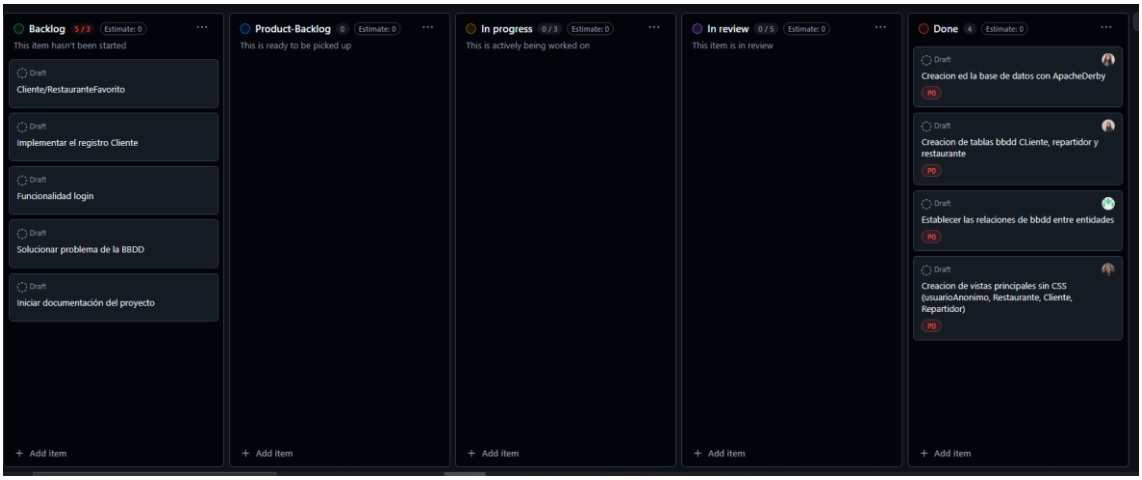


Ilustración 7: Backlog sprint 4

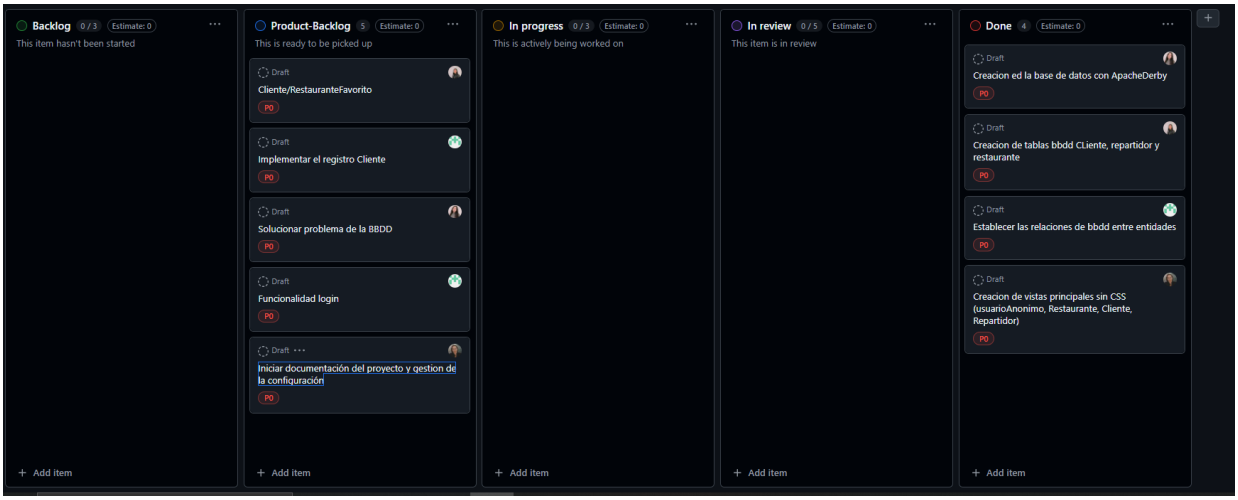


Ilustración 8: Product Backlog sprint 4

- **Sprint 5 (dos semanas):**
  - **Fechas:** 27/11/2024-11/12/2024

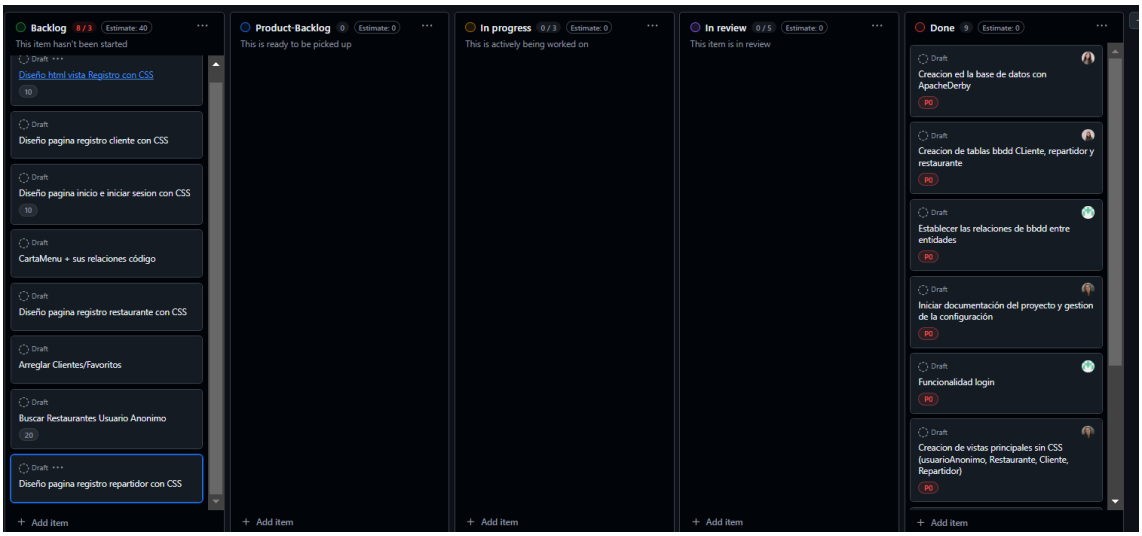


Ilustración 9: Backlog Sprint 5

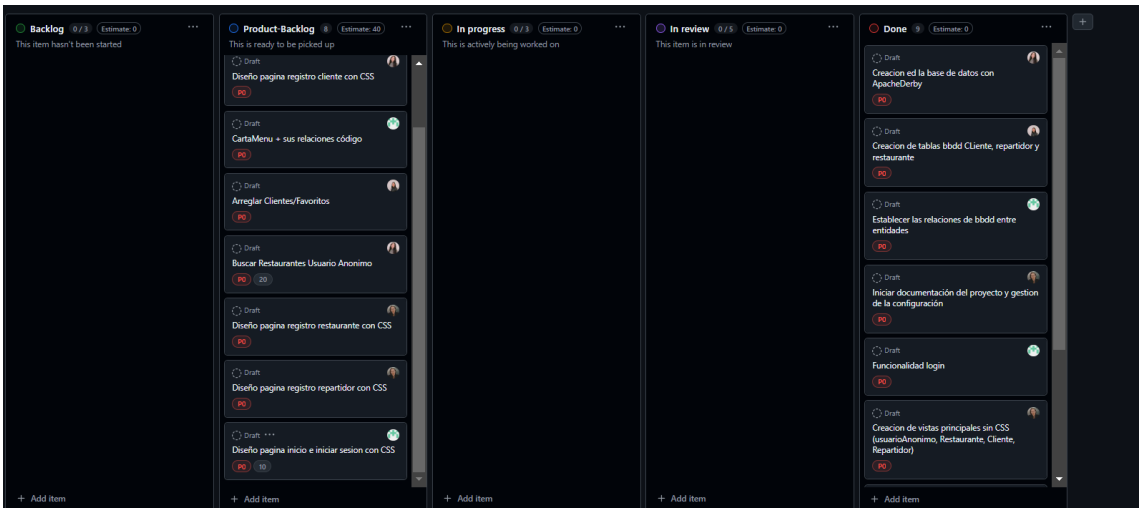
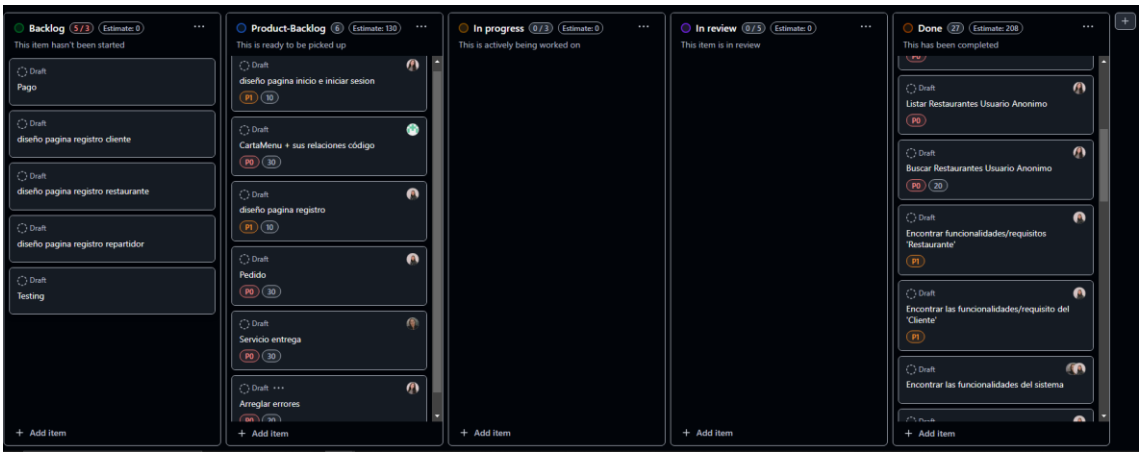
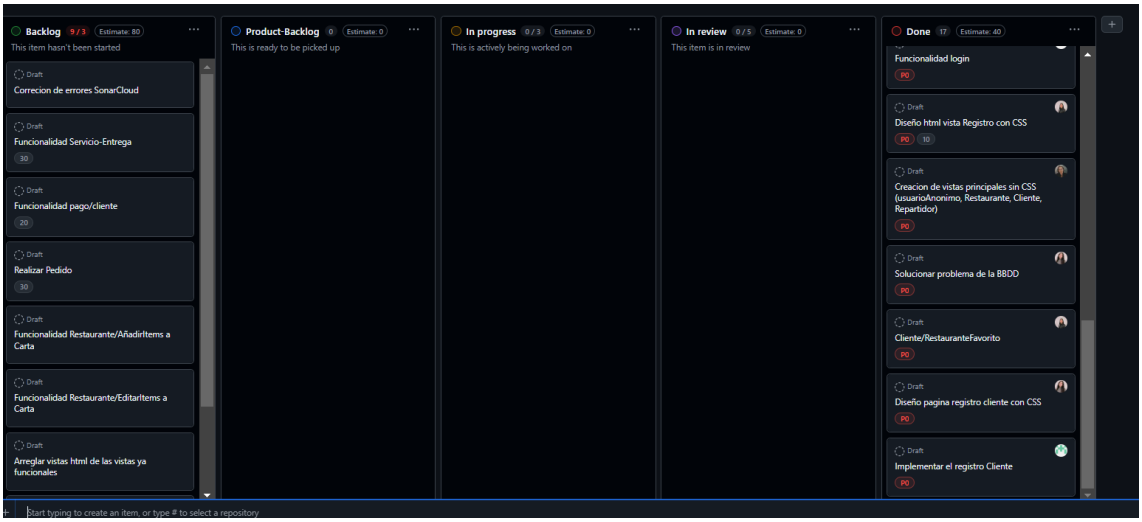


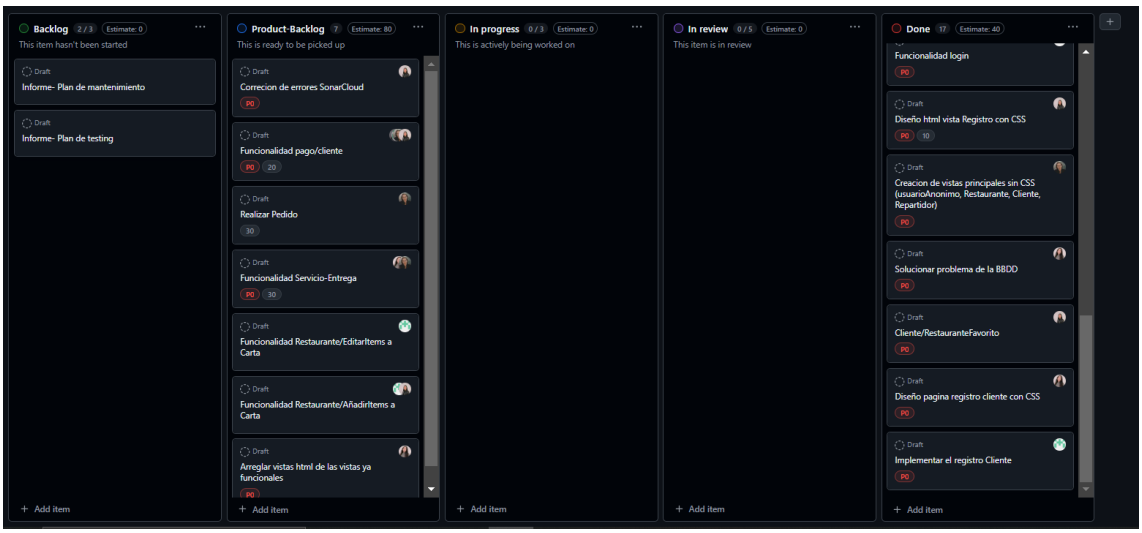
Ilustración 10: Product-Backlog Sprint 5

- **Sprint 6 (dos semanas):**
  - **Fechas**: 11/12/2024-25/12/2024



- **Sprint 7 (dos semanas):**
  - **Fechas:** 25/12/2024-8/01/2025





1.1 Plan de Iteraciones

Cada sprint comienza con una reunión que combina la revisión del sprint anterior y la planificación del próximo. En esta reunión, se presentan las tareas que han sido definidas en el backlog y han sido finalizadas, así como, las tareas que no se han podido finalizar. Esto nos ha permitido una retroalimentación constante sobre las tareas realizadas, nuestra organización, los problemas que han sucedido y de la carga de trabajo. Al mismo tiempo, se establecen los objetivos y se seleccionan las tareas a abordar en el siguiente sprint. Durante este proceso, todos los miembros del equipo comprenden las tareas asignadas y se definen los criterios de aceptación para cada historia de usuario, asegurando que todos estén alineados con las expectativas.

6. Implementación

6.1 Arquitectura del Sistema

El sistema sigue una arquitectura en capas, diseñada para separar las responsabilidades en el back-end de manera modular. Esta separación facilita el mantenimiento, la escalabilidad y la seguridad del sistema.

- **Capa de presentación**: Interactúa con la capa de negocio a través de una API, sirviendo como intermediario entre el cliente y el servidor.
- **Capa de lógica de negocio**: Implementa la lógica del negocio, gestionando el flujo de pedidos, la asignación de repartidores y el procesamiento de datos. Esta capa también se encarga de manejar las reglas de negocio y las validaciones.
- **Capa de datos**: Gestiona la persistencia de datos en la base de datos Derby. Incluye los modelos y consultas necesarias para la gestión de usuarios, pedidos y otros datos esenciales.

La base de datos en Apache Derby está estructurada para almacenar toda la información necesaria para el funcionamiento del sistema.

7. Plan de pruebas y de los Casos de pruebas

ENTITY

Clase Cliente

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
dni	String	Números del 0 al 9 y letras de la A a la Z	-	-	null o repetido
nombre	String	Conjunto de letras de la A a la Z	-	-	null
apellidos	String	Conjunto de letras de la A a la Z	-	-	null
contraseña	String	Números del 0 al 9, letras de la A a la Z o caracteres especiales	-	-	null
email	String	Conjunto de letras de la A a la Z o numeros del 0 al 9 con @gmail.com	-	-	null
dirección	Direccion	Conjunto direccion	-	-	null

Clase Restaurante

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
nombre	String	Conjunto de letras de la A a la Z	-	-	null
cif	String	Conjunto de letras de la A a la Z	-	-	null
passwordRestaurante	Strins	Números del 0 al 9, letras de la A a la Z o caracteres especiales	-	-	null
direccion	String	Números del 0 al 9, letras de la A a la Z o caracteres especiales	-	-	null

Clase Repartidor

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
dniRepartidor	String	Números del 0 al 9 y letras de la A a la Z	-	-	null
nombreRepartidor	String	Conjunto de letras de la A a la Z	-	-	null
apellidoRepartidor	String	Conjunto de letras de la A a la Z	-	-	null
passwordRepartidor	String	Números del 0 al 9, letras de la A a la Z o caracteres especiales	-	-	null
emailRepartidor	String	Conjunto de letras de la A a la Z o numeros del 0 al 9 con @gmail.com	-	-	null
disponible	boolean	True,False	-	-	-

Clase CartaMenu

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
nombreCarta	String	Números del 0 al 9, letras de la A a la Z o caracteres especiales	-	-	null
restaurante	Restaurante	Conjunto Restaurante	-	-	null

Clase Dirección

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
calle	String	Conjunto de letras de la A a la Z	-	-	null
numero	String	Números del 0 al 9	-	-	null
municipio	String	Conjunto de letras de la A a la Z	-	-	null
codigoPostal	Enumeracion	{45600, 28000, 45007}	-	-	null

Clase ItemMenu

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
nombre	String	Conjunto de letras de la A a la Z	-		null
precio	double	[0, infinito)	-	0	null o menor que cero
tipo	Enumeracion	{COMIDA,BEBIDA, POSTRE}	-	-	null
cartaMenu	CartaMenu	Conjunto CartaMenu	-	-	null

Clase Login

ATRIBUTOS DE PRUEBA	TIPO	CLASES DE EQUIVALENCIA	VALORES	VALORES LÍMITE	CONJETURA DE ERRORES
clienteDAO	ClienteDAO	-	-	-	null
restauranteDAO	RestauranteDAO	-	-	-	null
repartidorDAO	RepartidorDAO	-	-	-	null

CONTROLLER

GestorClientes

PARAMETROS	TIPOS	CLASES DE EQUIVALENCIA	CONJETURA DE ERRORES
registroCliente	String	Datos del cliente(dni, nombre, etc)	Introducir un cliente ya existente
mostrarFavoritos	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
agregarFavorito	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
eliminarFavorito	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
mostrarCliente	String	Datos del cliente(dni, nombre, etc)	Si no existen Clientes

GestorLogin

PARAMETROS	TIPOS	CLASES DE EQUIVALENCIA	CONJETURA DE ERRORES
mostrarLogin	String	Roles(Cliente, Restaurante, Repartidor)	Si se elije otro rol
logout	String	Sesión activa	Sesión no activa

GestorRepartos

PARAMETROS	TIPOS	CLASES DE EQUIVALENCIA	CONJETURA DE ERRORES
registroRepartidor	String	Datos del repartidor(dni, nombre, etc)	Introducir un repartidor ya existente o datos en formato incorrecto

GestorRestaurantes

PARAMETROS	TIPOS	CLASES DE EQUIVALENCIA	CONJETURA DE ERRORES
manejarMenuRestaurante	String	Datos del restaurante(cif, nombre, etc)	Introducir un cliente ya existente
registroRestaurante	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
vistaRestaurante	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
listarRestaurantesComunes	String	Datos Restaurante favorito(nombre,etc)	Si no existen Restaurantes favoritos
buscarRestaurantes	String	Datos del cliente(dni, nombre, etc)	Si no existen Clientes
verItemsDeCartaAnonimo	String	Datos de los items de la Carta(nombre,etc)	Si esa carta no tiene items
verItemsDeCartaUsuario	String	Datos de los items de la Carta(nombre,etc)	Si esa carta no tiene items
VerCartaMenu	String	Datos de las cartas(Nombre)	Si no existen Cartas
mostrarFormularioAgregarCartas	String	Datos de las cartas(Nombre)	Si no existen Cartas
mostrarFormularioEdicion	String	Datos de los items de la Carta(nombre,etc)	Introducir datos incorrectos(introducir una letra en el precio)
mostrarFormularioAgregarItem	String	Datos de los items de la Carta(nombre,etc)	Introducir datos incorrectos(introducir una letra en el precio)

8. Plan de Calidad de Software

Nos aparecían al principio un montón de errores que fuimos cambiando poco a poco.

En primer lugar, hemos tenido que definir varias constantes para no tener que ir escribiendo lo mismo todo el rato:

```
// constantes que necesitaremos
private static final String ERROR_RESTAURANTE_NO_ENCONTRADO = "Restaurante no encontrado";  8 usages
private static final String ERROR_CARTA_NO_ENCONTRADA = "Carta de menú no encontrada";  3 usages
private static final String ERROR_CARTA_NO_PERTENECE = "La carta de menú no pertenece al restaurante indicado.";
private static final String ERROR_ITEM_NO_ENCONTRADO = "Ítem no encontrado";  3 usages
public static final String STRING_RESTAURANTE = "restaurante";  15 usages
public static final String STRING_RESTAURANTES = "restaurantes";  2 usages
public static final String STRING_CARTAMENU = "cartaMenu";  10 usages
public static final String STRING_CARTASMENU = "cartasMenu";  2 usages
public static final String STRING_ITEMS = "items";  7 usages
public static final String STRING_CLIENTE = "cliente";  1 usage
private static final String VISTA_REGISTRO_RESTAURANTE = "registroRestaurante";  3 usages
private static final String VISTA_LISTAR_RESTAURANTES = "listarRestaurantes";  2 usages
private static final String REDIRECT_RESTAURANTE = "redirect:/restaurante/";  4 usages
public static final String STRING_ITEMS2 = "items[";  4 usages
```

También, tuvimos que crear un constructor en todos los gestores para eliminar el @Autowired:

```
// Constructor para poder eliminar el Autowired
public GestorRestaurantes(RestauranteDAO restauranteDAO, CartaMenuDAO cartaMenuDAO, ItemMenuDAO itemMenuDAO, ClienteDAO clienteDAO){
    this.restauranteDAO = restauranteDAO;
    this.cartaMenuDAO = cartaMenuDAO;
    this.itemMenuDAO = itemMenuDAO;
    this.clienteDAO = clienteDAO;
}
```

Luego, creamos excepciones personalizadas para saber cuál era el problema realmente:

```
class RestauranteNoEncontradoException extends RuntimeException {  10 usages
    public RestauranteNoEncontradoException(String message) {  10 usages  a
        super(message);
    }
}

class CartaNoEncontradaException extends RuntimeException {  3 usages  angel
    public CartaNoEncontradaException(String message) { super(message); }
}

class CartaNoPerteneceException extends RuntimeException {  1 usage  angelaa
    public CartaNoPerteneceException(String message) { super(message); }
}

class ItemNoEncontradoException extends RuntimeException {  3 usages  angela
    public ItemNoEncontradoException(String message) { super(message); }
}
```

9. Plan de Mantenimiento

Introducción

El objetivo de este plan es garantizar el correcto funcionamiento, actualización y optimización del proyecto, asegurando que se cumplan los requisitos del cliente y que se ofrezca una experiencia fluida tanto a los usuarios finales como a los administradores del sistema.

Tipos de Mantenimiento

Para asegurar el cumplimiento de los requisitos mencionados, se aplicarán los siguientes tipos de mantenimiento:

Mantenimiento Correctivo

- Reparación de errores en la funcionalidad (errores en el registro de pedidos, problemas con el sistema de pagos).
- Solución de problemas de rendimiento (lentitud en la carga de la página).

Mantenimiento Preventivo

- Actualización periódica de librerías, frameworks y dependencias.
- Limpieza de datos obsoletos (eliminar registros antiguos que ralentizan la base de datos).
- Realización de copias de seguridad semanales del sistema y la base de datos.

Mantenimiento Evolutivo

- Añadir nuevas funcionalidades requeridas por el cliente o los usuarios finales (opciones de pago adicionales, integración con nuevas plataformas de mensajería).

- Mejorar la interfaz de usuario (UI) y la experiencia de usuario (UX) para mantenerse competitivo en el mercado.

Mantenimiento Adaptativo

- Adaptar el sistema a cambios legales o regulatorios (normativas de protección de datos como el RGPD o nuevas leyes fiscales).
- Compatibilidad con nuevos navegadores o dispositivos.

Herramientas Utilizadas

Las herramientas seleccionadas para el desarrollo, testing y mantenimiento del proyecto son:

- **GitHub/GitLab**: Control de versiones y colaboración en el código.
- **Html:** Para el desarrollo de la interfaz de usuario.
- **Spring Boot (Java)**: Framework para el backend.
- **Maven: G**estión y construcción del proyecto de software
- **Derby Embebido**: Sistema de gestión de bases de datos.
- **SonarCloud**: Análisis de calidad del código.

10. Contribución

HOUDA EL OUAHABI KHIRAT (Houda.Elouahabi@alu.uclm.es)

ÁNGELA JIAO PENG (Angela.Jiao@alu.uclm.es)

SARA GUTIÉRREZ CAJA (Sara.Gutierrez4@alu.uclm.es)

INGRID NIVEIRO BEN (Ingrid.Niveiro@alu.uclm.es)