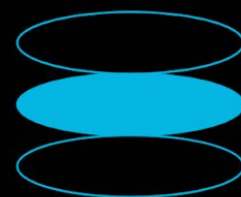




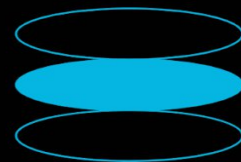
# COMMERCEBOOST

Potencia tu negocio



## Indice

• Índice .....	1
• ¿Que es CommerceBoost? .....	2
• Objetivo .....	2
▪       Objetivo general .....	2
•       Objetivo específico .....	2
• Estudio de mercado .....	3
• Análisis de la competencia .....	3
• Presupuesto .....	3
• Análisis y especificación de requisitos .....	4
• Requisitos funcionales .....	4
• Manual de instalación .....	7
• Dependencias .....	7
• .....	



## ¿Que es CommerceBoost?

CommerceBoost es una aplicación de sistema de Punto de Venta Profesional (TPV) que permite gestionar ventas, inventario y caja de forma sencilla y eficiente. Se diseñó para optimizar el día a día de comercios con una interfaz rápida e intuitiva.

Nos centramos en modernizar la gestión de pequeñas y medianas empresas, haciendo que estas puedan administrar ventas, inventario, y cierres de caja de una manera sencilla e intuitiva.

Debido a que la digitalización es un paso que no se va a poder evitar decidí implementar un sistema de gestión fácil e intuitivo, pensado para personas sin mucho conocimiento.

Todo surgió al trabajar en la tienda de mi madre. La aplicación de TPV que usaba era bastante antigua y enrevesada para ella, por lo que decidí hacer una interfaz más visual y no muy cargada. Con las interacciones justas y necesarias. Otro punto que ha influido en la estructura de CommerceBoost, fue la necesidad de reducir la pérdida de tiempo en pantallas (Opciones de cobro directo en un único click).

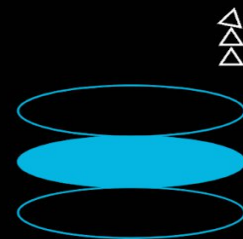
## Objetivo

### Objetivo general

Desarrollar un sistema TPV de escritorio robusto y escalable que cubra el ciclo completo de venta presencial, desde la selección de productos hasta la emisión del ticket y el cierre de caja. Todo de la manera más intuitiva y sencilla posible..

### Objetivo específico

- **Gestión de Ventas:** Implementar un flujo de venta rápido (menos de 3 clics por artículo) con soporte para escáner de códigos de barras.
- **Control de Inventario:** Facilitar el mantenimiento del catálogo de productos en tiempo real.
- **Flexibilidad de Cobro:** Soportar múltiples métodos de pago (Efectivo, Tarjeta) y descuentos.
- **Fiabilidad Contable:** Garantizar la integridad de los datos de ventas y generar reportes de cierre de caja exactos.
- **Experiencia de Usuario:** Crear una interfaz intuitiva que no requiera formación técnica para su uso.



## Estudio de mercado

Actualmente, el mercado ofrece dos tipos de soluciones:

- **Software Legacy:** Programas antiguos, caros y difíciles de usar.
- **SaaS (Nube):** Modernos pero requieren pago mensual (suscripción).

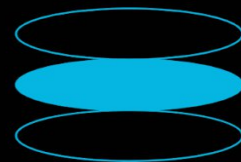
**CommerceBoost** se posiciona como una alternativa intermedia: un software moderno, fácil de usar, pero de pago único y que funciona en local sin depender de internet, ideal para pequeños comercios que quieren evitar costes fijos mensuales.

### Análisis de la competencia

Solución	Ventajas	Desventajas
Caja Registradora	Barata, robusta	Sin control de stock, sin datos
TPV Bancario	Integrado con pagos	Funcionalidad muy limitada
Software Propietario	Completo	Coste licencia (>500€), soporte caro
<b>CommerceBoost</b>	<b>Sin cuotas, control total, UX moderna</b>	Requiere hardware PC (Windows)

## Presupuesto

Concepto	Unidades	Coste Unitario	Total
<b>Recursos Humanos</b>			
Desarrollador Junior	100 horas	20 €/h	2.000 €
Diseñador UI/UX	20 horas	25 €/h	500 €
<b>Hardware y Software</b>			
Licencia Windows 11 Pro	1	145 €	145 €
Amortización Equipo	1	200 €	200 €
<b>Total Estimado</b>			<b>2.845 €</b>



## Análisis y especificación de requisitos

### Requisitos funcionales

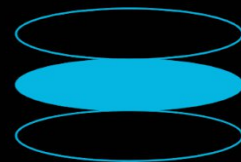
- **Gestión de ventas:**
  - Añadir productos, modificar cantidades y calcular totales.
- **Inventario CRUD:**
  - Crear.Leer.Actualizar y Borrar productos del catalogo.
- **Persistencia:**
  - Procesar pagos en Efectivo/Tarjeta y generar ticket
- **Cierre Z:**
  - Generar informe diario de ventas y reiniciar caja.

### Requisitos no funcionales

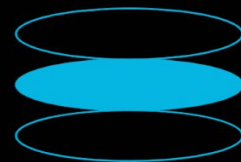
- **Usabilidad:**
  - Interfaz intuitiva con botones grandes.
- **Rendimiento:**
  - Respuesta inmediata.
- **Persistencia:**
  - Base de datos local PostgreSQL
- **Seguridad:**
  - Datos almacenados localmente.

### Casos de uso

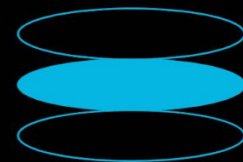
- **Realizar venta con efectivo:**
  - **Usuario:** Cajero.
  - **Precondicion:** Sistema iniciado, productos en inventario.
  - **Flujo Principal:**
    1. Cajero escanea producto.
    2. Sistema añade profucto a lista de venta.
    3. Cajero modifica cantidad si es necesario (numpad)
    4. Cajero repite pasos 1-3 para mas productos.
    5. Cajero presiona "Cobrar".
    6. Sistema genera ticker y lo guarda.
    7. Sistema resetea venta actual
  - **Flujo alternativo:**
    1. Cajero selecciona articulo
    2. Cajero presiona "Descuento".
    3. Cajero introduce porcentaje.



4. Sistema aplica descuento y recalcula el precio.
- **Realizar cierre de caja:**
  - **Usuario:** Encargado.
  - **Precondición:** Sistema iniciado, productos en inventario.
  - **Flujo Principal:**
    1. Usuario presiona "Ventas".
    2. Sistema muestra un resumen diario.
    3. Usuario presiona "Imprimir Cierre Z"
    4. Sistema genera reporte con:
      - Desglose por metodo de pago.
      - Lista de cada venta.
      - Totales generales.
    5. Sistema marca ventas como cerradas
    6. Sistema resetea contadores a 0.
- **Modificar precio de articulo:**
  - **Usuario:** Cajero.
  - **Precondición:** Artículo añadido a venta.
  - **Flujo principal:**
    1. Cajero hace clic en columna "PRECIO UNIT." del artículo.
    2. Sistema cambia modo de edición a precio.
    3. Cajero introduce el nuevo precio con el numpad
    4. Sistema actualiza precio y recalcula total.
- **Consultar Historial de Ventas:**
  - **Usuario:** Encargado.
  - **Precondición:** Existen ventas registradas.
  - **Filtro Principal:**
    1. Usuario presiona "Ventas"
    2. Sistema muestra pestaña "Resumen Diario".
    3. Usuario puede ver:
      - Total ventas del dia
      - Lista de tickets.
    4. Usuario cambia a pestaña "Historial de Tickets".
    5. Sistema muestra lista completa de ventas.
- **Realizar venta con Tarjeta:**
  - **Usuario:** Cajero.
  - **Precondición:** Sistema iniciado, productos en inventario.
  - **Flujo Principal:**
    1. Cajero escanea productos.
    2. Cajero presiona "Tarjeta"
    3. Sistema registra método de pago como "Tarjeta".
    4. Sistema genera ticket indicando el pago con tarjeta.



5. Sistema resetea venta actual.
- **Venta sin ticket:**
    - **Usuario:** Cajero.
    - **Precondición:** Sistema iniciado, productos en inventario.
    - **Filtro principal:**
      1. Cajero escanea productos.
      2. Cajero presiona "Sin ticket".
      3. Sistema registra venta en base de datos.
      4. Sistema NO genera ticket.
      5. Sistema resetea venta actual.
  - **Reimprimir Último Ticket:**
    - **Usuario:** Cajero.
    - **Precondición:** Se ha realizado al menos una venta.
    - **Flujo Principal:**
      1. Cajero presiona "Reimprimir".
      2. Sistema recupera ultima venta registrada.
      3. Sistema genera copia del ticket.
      4. Sistema guarda ticket con sufijo "\_REIMPRESION"
  - **Abrir cajón de dinero:**
    - **Usuario:** Cajero.
    - **Precondición:** Sistema iniciado.
    - **Flujo principal:**
      1. Cajero presiona boton "Cajon".
      2. Sistema procede a la apertura de cajon.
  - **Eliminar artículo de venta:**
    - **Usuario:** Cajero.
    - **Precondición:** Hay articulos en la venta actual.
    - **Flujo Principal:**
      1. Cajero selecciona artículo en lista.
      2. Cajero presiona "Borrar".
      3. Sistema elimina artículo de la venta (si no selecciono articulo elimina el último que se añadió)
      4. Sistema recalcula total automaticamente.



## Manual de instalación

### Dependencias

- Windows 10/11
- .NET 8.0 SDK
- Docker Desktop
- Git

### Instalacion paso a paso

- **Paso 1: Instalar .NET 8.0 SDK**

- Descargar e instalar desde: <https://dotnet.microsoft.com/download/dotnet/8.0>
- Verificar instalación:

```
dotnet --version
```

- Debe mostrar una version 8.0.x

- **Paso 2: Instalar Docker Desktop**

- Descargar e instalar Docker Desktop desde: <https://www.docker.com/products/docker-desktop> . Una vez instalado, iniciar Docker Desktop y verificar que esta corriendo.

- **Paso 3: Clonar el repositorio**

```
git clone https://github.com/davidfernandeztorres/CommerceBoost.git
```

```
cd CommerceBoost
```

- **Paso 4: Levantar la base de datos con Docker Compose**

- Ejecutar el siguiente comando en la raíz del proyecto:

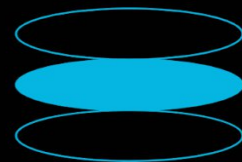
```
docker compose up -d
```

- Este comando :

1. Levanta PostgreSQL en el puerto 5433
2. Levanta pgAdmin en <http://localhost:8080>
3. Crea automaticamente la base de datos "commerce"
4. Configura usuario: postgres, contraseña: password

- **Paso 5: Verificar que los contenedores estan corriendo**

```
docker compose ps
```



- **Paso 6: Restaurar paquetes NuGet**

`Dotnet restore`

- **Paso 7: Aplicar migraciones a la base de datos**

- Instalar la herramienta de Entity Framework:

`dotnet tool install --global dotnet-ef`

- Crear las tablas en PostgreSQL:

`dotnet ef database update`

- **Paso 8: compilar el proyecto**

`dotnet build`

- **Paso 9: Ejecutar la aplicación**

`dotnet run`

La aplicación se abrirá y estará lista para usar

## Solución de problemas comunes

- **Error: "No se puede conectar a la base de datos"**

- Verificar que Docker Desktop está corriendo
- Verificar que los contenedores están activos: `docker compose ps`
- Si no están activos, levantarlos: `docker compose up -d`

- **Error: "El puerto 5433 ya está en uso"**

- Hay otra instancia de PostgreSQL corriendo
- Cambiar el puerto en `docker-compose.yml` o detener el servicio conflictivo

- **Error: "dotnet-ef no se reconoce"**

- Instalar la herramienta: `dotnet tool install --global dotnet-ef`
- Reiniciar la terminal después de instalar





# COMMERCEBOOST

Potencia tu negocio

