1. Cálculos y análisis realizados

1.1 Promedios (medias)

Promedio de ventas por día

```
SELECT fecha::date AS dia, AVG(total_venta) AS promedio_venta_dia FROM ventas
GROUP BY fecha::date ORDER BY dia;
```

Promedio de unidades vendidas por producto

```
SELECT p.nombre, AVG(d.cantidad) AS promedio_unidades FROM detalle_ventas d
JOIN productos p ON p.id_producto = d.id_producto GROUP BY p.nombre;
```

Promedio de ventas por cliente

```
SELECT c.nombre, AVG(v.total_venta) AS promedio_cliente FROM ventas v JOIN
clientes c ON c.id_cliente = v.id_cliente GROUP BY c.nombre;
```

1.2 Desvío estándar

Variabilidad del monto diario

```
SELECT fecha::date AS dia, STDDEV(total_venta) AS variabilidad_diaria FROM
ventas GROUP BY dia;
```

Variabilidad por método de pago

```
SELECT metodo_pago, STDDEV(total_venta) AS variabilidad FROM ventas GROUP BY
metodo_pago;
```

Interpretación típica:

Valores altos indican ventas muy irregulares (picos y valles); valores bajos indican comportamiento estable y predecible.

1.3 Correlaciones

1.3.1 Precio ↔ Cantidad vendida

Datos necesarios: tabla agregada por producto.

```
SELECT p.precio, SUM(d.cantidad) AS cantidad_total FROM productos p JOIN
detalle_ventas d ON d.id_producto = p.id_producto GROUP BY p.id_producto;
```

Coeficiente de Pearson:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Interpretación:

- **Cercano a -1:** Elasticidad negativa (productos más caros venden menos unidades).
- **0:** No hay relación significativa.
- **Cercano a +1:** Anomalía (productos más caros venden más, posible bien de lujo).

1.3.2 Cantidad vendida ↔ Día de la semana

```
SELECT EXTRACT(DOW FROM v.fecha) AS dia_semana, SUM(d.cantidad) AS cantidad FROM ventas v JOIN detalle_ventas d ON d.id_venta = v.id_venta GROUP BY dia_semana ORDER BY dia_semana;
```

Nota: Convertir día en valores 0–6 para correlación numérica.

1.3.3 Monto total ↔ Método de pago

Dado que el método de pago es una variable categórica, analizamos la relación comparando el **Ticket Promedio** por medio de pago para detectar si ciertos métodos incentivan compras mayores.

```
SELECT metodo_pago, AVG(total_venta) AS ticket_promedio FROM ventas GROUP BY metodo_pago ORDER BY ticket_promedio DESC;
```

Interpretación:

Si el promedio de Tarjeta es significativamente mayor al de Efectivo, indica que la liquidez del cliente limita el volumen de compra en efectivo.

1.4 Visualizaciones recomendadas

Ventas promedio por día (Gráfico de Líneas)

- **Eje X:** Días del mes.
- **Eje Y:** Monto total vendido.
- **Objetivo:** Identificar tendencias temporales, estacionalidad semanal o el impacto de promociones en días específicos.

Relación precio–cantidad (Gráfico de Dispersión / Scatter Plot)

- **Eje X:** Precio del producto.
- **Eje Y:** Cantidad total vendida.

- **Objetivo:** Visualizar la elasticidad de la demanda. Se espera una curva descendente (a mayor precio, menor cantidad). Los puntos fuera de la curva (outliers) son productos de alto rendimiento o bajo interés.

Ventas por método de pago (Gráfico de Barras)

- **Eje X:** Método de pago (Tarjeta, Transferencia, Efectivo).
- **Eje Y:** Cantidad de transacciones y Monto acumulado.
- **Objetivo:** Comparar no solo cuánto dinero ingresa por cada canal, sino la frecuencia de uso de cada uno.

---

2. Informe Técnico

2.1 Estructura de la base de datos

2.2 Tablas principales

Tabla	Descripción
`clientes`	Información básica de contacto de los clientes.
`productos`	Catálogo de ítems con precio, stock y categorización básica.
`ventas`	Cabecera de la transacción (quién compró, cuándo y cuánto pagó).
`detalle_ventas`	Desglose de ítems (qué productos específicos se incluyeron en cada venta).

2.3 Relaciones

- `ventas.id_cliente` → `clientes.id_cliente`
- `detalle_ventas.id_venta` → `ventas.id_venta`
- `detalle_ventas.id_producto` → `productos.id_producto`

Notas Técnicas

- `subtotal` en `detalle_ventas` se define como columna generada: `GENERATED ALWAYS AS (cantidad * precio_unitario) STORED`.
- `total_venta` en `ventas` requiere actualización posterior a la inserción de los detalles (vía Trigger o UPDATE manual en transacción).

2.4 Datos iniciales (Seed Data)

- **Clientes:** 30 registros.
- **Productos:** 20 registros, divididos en categorías:
  - 1–5: Snacks
  - 6–9: Bebidas
  - 10–13: Limpieza/Hogar
  - 14–17: Alimentos frescos
  - 18–20: Saludables
- **Ventas:** 60 registros generados entre el 01/11/2025 y el 07/11/2025.
- **Volumen:** Entre 1 y 6 productos por venta.
- **Métodos de pago:** Distribución aproximada: 50% tarjeta, 35% transferencia, 15% efectivo.

Definición de Esquemas

Tabla: `clientes`

Campo	Tipo
id_cliente	int PK
nombre	varchar
email	varchar
telefono	varchar
activo	boolean

Tabla: `productos`

Campo	Tipo
id_producto	int PK
nombre	varchar
categoria	varchar
marca	varchar
precio	numeric
stock	int
descripcion	text
activo	boolean

Tabla: `ventas`

Campo	Tipo
id_venta	int PK
fecha	timestamp
id_cliente	int FK
metodo_pago	varchar
total_venta	numeric

Tabla: `detalle_ventas`

Campo	Tipo
id_detalle	int PK
id_venta	int FK
id_producto	int FK
cantidad	int
precio_unitario	numeric

2.5 Consultas SQL principales

Listado de ventas con totales

```
SELECT v.id_venta, v.fecha, c.nombre AS cliente, v.metodo_pago, v.total_venta
FROM ventas v JOIN clientes c ON c.id_cliente = v.id_cliente;
```

Detalle completo de una venta

Nota: Aunque el subtotal está almacenado, se puede recalcular para validación.

```
SELECT d.id_producto, p.nombre, d.cantidad, d.precio_unitario, (d.cantidad *
d.precio_unitario) AS subtotal_calculado FROM detalle_ventas d JOIN productos p
ON p.id_producto = d.id_producto WHERE d.id_venta = $1;
```

Unidades vendidas por categoría

```
SELECT p.categoria, SUM(d.cantidad) AS unidades_vendidas FROM productos p JOIN
detalle_ventas d ON d.id_producto = p.id_producto GROUP BY p.categoria;
```

Ventas por día y método de pago

```
SELECT fecha::date AS fecha, metodo_pago, SUM(total_venta) AS total FROM ventas
GROUP BY fecha::date, metodo_pago ORDER BY fecha, metodo_pago;
```

2.6 Variables analizadas y significado

Variable	Significado
`precio`	Precio unitario de lista del producto.
`cantidad`	Número de unidades de un mismo producto en una venta.
`total_venta`	Suma de todos los subtotales de la transacción.
`metodo_pago`	Medio utilizado (Efectivo, Tarjeta, Transferencia).
`categoria`	Clasificación del producto para agrupar métricas.

2.7 Resultados esperados de análisis

- **Rotación de Stock:** Las categorías "Snacks" y "Bebidas" suelen mostrar mayor rotación en volumen (cantidad), aunque no necesariamente en margen de ganancia.
- **Sensibilidad al Precio:** Se espera una correlación negativa moderada entre precio y cantidad; los productos de limpieza (compra planificada) pueden ser menos sensibles que los snacks (compra impulsiva).
- **Patrón de Pagos:** Transferencias y tarjetas tenderán a asociarse con tickets promedio más altos en comparación con el efectivo.
- **Estabilidad:** El desvío estándar ayudará a identificar si las ventas son consistentes o si dependen de "golpes de suerte" en días específicos.

---

3. Endpoints de la API

3.1. CRUD — Clientes

Método	Ruta	Descripción
GET	`/clientes`	Listar todos los clientes activos.

3.2. CRUD — Productos

Método	Ruta	Descripción
GET	`/productos`	Listar catálogo de productos.

3.3. CRUD — Ventas

Método	Ruta	Descripción
GET	`/ventas`	Listar historial de ventas.
POST	`/ventas`	Registrar nueva venta (incluye creación de detalles).
PUT	`/ventas/:id`	Editar cabecera de venta.
DELETE	`/ventas/:id`	Anular venta (Soft delete o borrado en cascada).

3.4. CRUD — Detalle de ventas

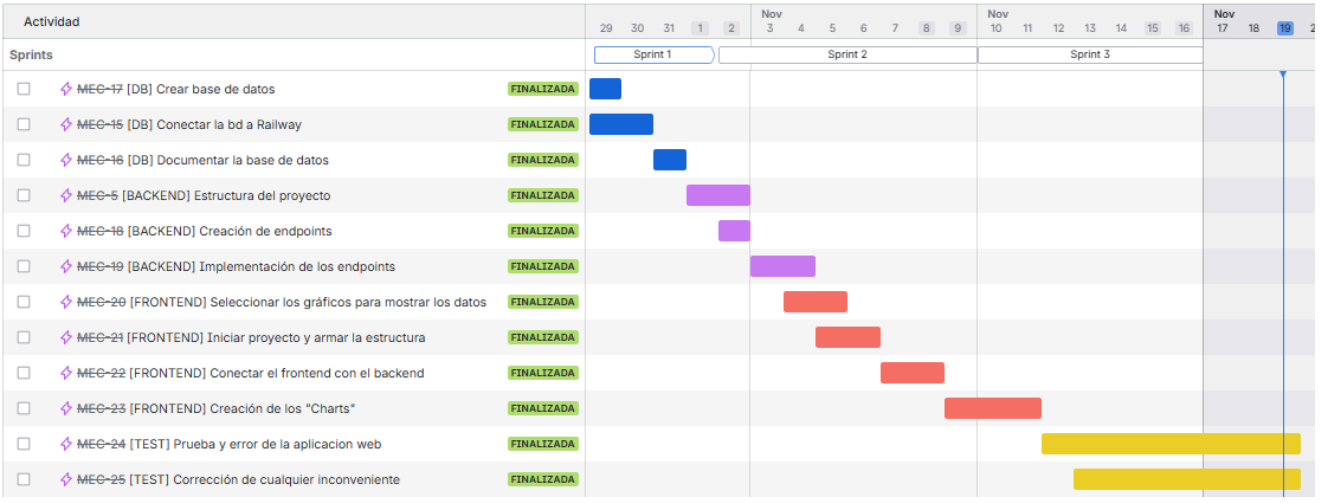
Método	Ruta	Descripción
GET	`/ventas/:id/detalle`	Obtener los productos asociados a una venta específica.

---

4. Estimación de Tareas y Tiempos

4.1 Planificación del Proyecto

Usamos Jira para planear el proyecto, dividiéndolo en “Sprints”. Esta herramienta de organización de trabajo cooperativo nos permitió planificar el curso de desarrollo de cada uno de los integrantes del grupo de una manera eficiente. Consideramos que la experiencia en la plataforma fue satisfactoria.



4.2 Comienzo del proyecto

4.2.1 Base de Datos

Tatiana Roque se encargó de la creación y conexión de la base de datos. Se crearon todas las tablas con sus datos correspondientes, ya pensando en una estructura sólida y coherente para que el proyecto sea escalable y no haya problemas a futuro. Al terminar la base de datos se la conectó con Railway siendo así accesible para el resto del equipo.

4.2.2 Estructura del Backend

Una vez que la base de datos estaba funcionando, Tatiana compartió los endpoints. Ezequiel Bergamini inició la estructura del backend, vinculandolo con Railway. La estructura de la API de se dividió en Model, Controllers, Routes y Server.

4.3 Intermedio del proyecto

4.3.1 Backend funcional

Ezequiel Bergamini implementó la lógica de negocio utilizando Node.js con Express, empleando TypeScript para asegurar la robustez y tipado del código. Se integró Sequelize como ORM para la interacción con la base de datos de Railway, mapeando los modelos clientes, productos, ventas y detalle_ventas. Se desarrollaron todos los controllers y routes necesarios para los endpoints CRUD definidos en la sección 3 (Clientes, Productos, Ventas, Detalle de Ventas), asegurando la correcta manipulación de datos y respuestas JSON estandarizadas.

4.3.2 Comienzo del Frontend

Ariel tuvo la responsabilidad de diseñar la interfaz de usuario. Su trabajo se inició con la fase de diseño, donde se utilizó la herramienta colaborativa Figma para definir con precisión cada elemento visual y la experiencia de usuario general del front-end. Una vez finalizado y aprobado el diseño en Figma, se procedió a la etapa de implementación. Para ello, se optó por la biblioteca de JavaScript React, lo que les permitió construir la interfaz de manera modular, eficiente y con un alto nivel de interactividad, asegurando que el producto final fuera tanto visualmente atractivo como funcionalmente robusto.

4.3.3 Conexión Frontend con el Backend

Ezequiel, Ariel y Tatiana tomaron la tarea de establecer la conexión y garantizar la comunicación fluida entre la API y el lado de la Interfaz de Usuario (frontend). Nuestro trabajo se centró en asegurar que los datos procesados y ofrecidos por la API fueran correctamente solicitados, recibidos e interpretados para ser presentados de manera efectiva y usable dentro del entorno visual de la aplicación. Esto requirió una gran coordinación como equipo y un entendimiento tanto de la lógica de negocio detrás de la API como de las tecnologías y el estado de gestión de la interfaz de usuario.

4.4 Final del proyecto

4.4.1 Charts incorporados

Ariel desarrolló gráficos para la visualización de datos. Estos habían sido previamente implementados de manera preventiva y precisaban de un pulido final para poder ser óptimos para mostrarse.

4.5 Pruebas finales

4.5.1 Testeos

Ezequiel, Ariel y Tatiana realizaron pruebas de rendimiento a la aplicación, con el propósito de comprobar el óptimo funcionamiento de los repositorios entre sí y con la base de datos.

4.5.2 Corrección de errores inesperados

Tatiana se encargó de evaluar y corregir los errores finales de conexión entre la base de datos y la aplicación desplegada. Estos errores en gran parte surgieron al momento de actualizar los datos propios de las tablas al optar por utilizar nuevas muestras mock para los estadísticos.

---

5. Compilación del programa

5.1. Requisitos previos

- Node.js (v18 o superior)
- NPM

5.2. Configuración del Backend

1. Crear el archivo `.env` en la carpeta del backend:

```
PORT=3001
DB_HOST=shortline.proxy.rlwy.net
DB_PORT=19091
DB_USER=root
DB_PASSWORD=jwCZfIYwUlzYldbqsEJsSkiUFPxbaaOS
DB_NAME=mercado_estadisticas
DB_DIALECT=mysql
```

2. Instalar dependencias:
   `npm install`
3. Ejecutar el backend en modo desarrollo:
   `npm run dev`

El servidor queda disponible en:
`http://localhost:3001`

5.3. Configuración del Frontend

1. Instalar dependencias:
   `npm install`
2. Ejecutar la aplicación:
   `npm run dev`

La aplicación se servirá localmente (`http://localhost:3001`).