







Apertura: miércoles, 22 de octubre de 2025, 00:00

Cierre: miércoles, 5 de noviembre de 2025, 23:59

Trabajo Práctico 3: Simulador de Pedidos de Cafetería (TDD + React Testing Library + MSW)

Tabla de contenidos

-  Objetivo Central
-  Stack Tecnológico
-  Contexto del Proyecto
-  Parte I — Configuración del entorno de pruebas
-  Parte II — Desarrollo Guiado por Pruebas
-  Parte III — Integración Completa



Objetivo Central

Aplicar **Desarrollo Guiado por Pruebas (TDD)** en React para construir una pequeña aplicación que simule el flujo de pedidos en una cafetería. El estudiante deberá demostrar dominio del ciclo **Rojo → Verde → Refactor**, el uso de **React Testing Library (RTL)** con consultas accesibles, el manejo de estado y la integración con una **API simulada mediante MSW**.



Stack Tecnológico

- React + TypeScript + Vite
- Vitest + React Testing Library + @testing-library/user-event
- MSW (Mock Service Worker)
- Zod para validaciones
- Context API o hooks personalizados para el estado global



Contexto del Proyecto

Una cafetería desea digitalizar el proceso de toma de pedidos. El sistema mostrará un **menú de productos**, permitirá **agregar ítems a un pedido**, **visualizar el total a pagar**, y **enviar el pedido** (simulado). El objetivo del trabajo es implementar progresivamente cada funcionalidad **siguiendo la metodología TDD**, escribiendo primero los tests y luego la mínima implementación que los haga pasar.



Parte I — Configuración del entorno de pruebas

1. Crear el proyecto base:

```
npm create vite@latest cafeteria -- --template react-ts
```

2. Instalar dependencias de testing:

```
npm i -D vitest @testing-library/react @testing-library/user-event @testing-library/jest-dom msw zod
```

3. Configurar `vite.config.ts` :

```
test: {
  environment: 'jsdom',
  globals: true,
  setupFiles: './src/setupTests.ts',
}
```

4. Configurar `setupTests.ts` :

```
import '@testing-library/jest-dom';
import { server } from './mocks/server';
beforeAll(() => server.listen());
afterEach(() => server.resetHandlers());
afterAll(() => server.close());
```



Parte II — Desarrollo Guiado por Pruebas



Tipado base

Definir tipo y esquema:

```
const ProductSchema = z.object({
  id: z.string(),
  name: z.string().min(2),
  price: z.number().positive(),
});
type Product = z.infer<typeof ProductSchema>;
```

◆ HU1 — Visualización inicial del menú

Como usuario , quiero ver un listado de productos disponibles cuando ingreso al sistema, para poder elegir qué pedir.

- **Rojo:** test que verifique que se muestran productos mockeados por la API (`screen.getByText('Café')`).
- **Verde:** implementar fetch a `/api/menu` (interceptado por MSW).
- **Refactor:** separar el componente `<Menu />` .

👉 Testear con `await waitFor(...)` y `screen.getAllByRole('listitem')` .

◆ HU2 — Agregar ítem al pedido

Como usuario , quiero agregar productos al pedido, para calcular el total.

- Test: simular `click` sobre el botón “Agregar” de un producto.
- Verificar que aparece en el área de pedido (`getByRole('list')`).
- Implementar estado local o contexto (`useOrder`).

◆ HU3 — Calcular total del pedido

Como usuario , quiero ver el total actualizado cada vez que agrego o elimino un producto.

- Test: agregar varios productos y verificar el texto `"Total: $..."` .
- Implementar cálculo dinámico.
- Validar con `expect(screen.getByText(/total: \$\d+/i)).toBeInTheDocument()` .

◆ HU4 — Eliminar ítem del pedido

Como usuario , quiero poder quitar un ítem del pedido sin borrar todo.

- Test: verificar que el clic en “Eliminar” remueve solo ese producto.
- Implementar `e.stopPropagation()` si se anidan botones.
- Usar `setState` funcional.

◆ HU5 — Enviar pedido (MSW + Contexto)

Como usuario , quiero enviar mi pedido al servidor para confirmarlo.

- Mockear endpoint `/api/orders` con MSW .
- Test:
 1. Agregar varios ítems.
 2. Click en “Enviar pedido”.
 3. Esperar `await waitFor(...)` que muestre mensaje “Pedido confirmado”. * Implementar envío y limpiar estado tras éxito.

◆ HU6 — Caso límite: error o menú vacío

- Usar `server.use()` para simular un error 500 o lista vacía.
- Verificar que la app muestre “No hay productos disponibles” o “Error al cargar menú”.



Parte III — Integración Completa

Tests que cubran el flujo completo:

1. Cargar menú (mock).

- Agregar entrega

Estado de la entrega	Todavía no se han realizado envíos
Estado de la calificación	Sin calificar
Tiempo restante	12 días 7 horas restante
Última modificación	-
Comentarios de la entrega	▶ Comentarios (0)