DOCUMENTACIÓN DE ENTREGA Y TESTING
Proyecto: Dulce Vida Web

Autor: Ignacio Silva
Institución: DUOC UC – Escuela de Informática y Telecomunicaciones
Fecha: 24/11/2025

Índice

1. Documentación de la Entrega

Documentación de Testing y API - Dulce Vida Web

Nombre: Ignacio Silva

Fecha: 24 de noviembre de 2025

---

1. Introducción

El presente documento detalla la estrategia de pruebas (Testing) y la documentación de la API (Swagger) implementada para el proyecto Dulce Vida Web. El objetivo es validar el correcto funcionamiento de los requisitos funcionales críticos (autenticación, carrito, boleta) y asegurar la calidad del código tanto en el frontend como en el backend.

---

2. Estrategia de Testing

2.1 Herramientas Utilizadas

- Frontend: Vitest (framework de pruebas), React Testing Library (renderizado y aserciones de componentes), JSDOM (entorno de navegador simulado).

- Backend: JUnit 5 (pruebas unitarias), Mockito (simulación de dependencias), Spring Boot Test.

2.2 Plan de Pruebas

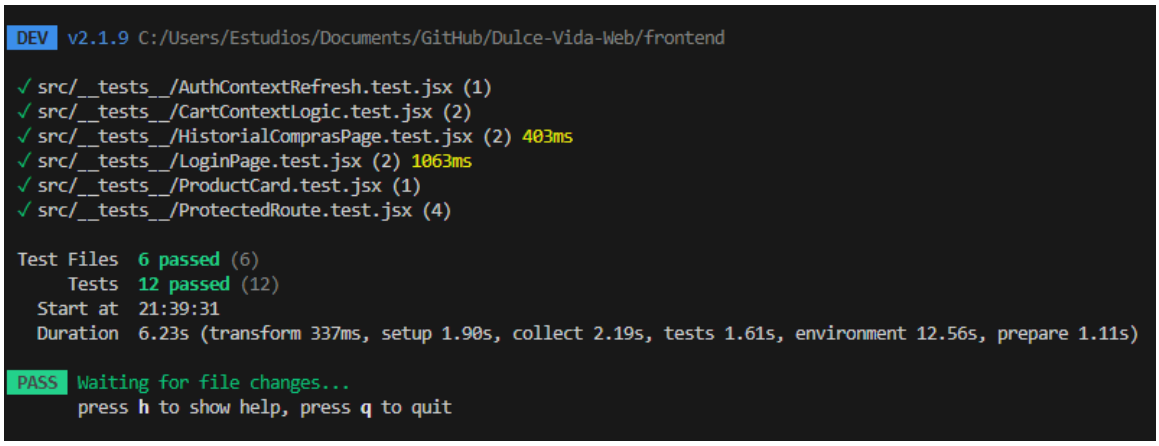
Se han seleccionado componentes críticos para asegurar la cobertura de los flujos principales:

1. Autenticación: Verificación de login, manejo de tokens y protección de rutas (`ProtectedRoute`).
2. Carrito de Compras: Lógica de negocio del carrito (agregar, totalizar) y persistencia de estado.
3. Interfaz de Usuario: Renderizado correcto de productos (`ProductCard`) y páginas principales (`HistorialComprasPage`).

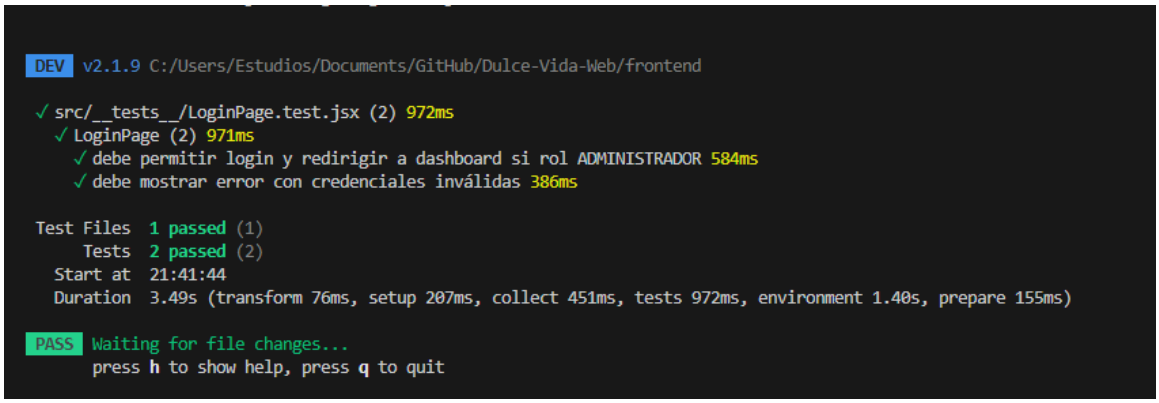
2.3 Resultados de Ejecución (Frontend)

Se ejecutó la suite de pruebas completa mediante el comando `npm test`.

Resultado: 6 archivos de prueba exitosos, 12 tests pasados.



> Figura 1: Ejecución exitosa de tests en Frontend.

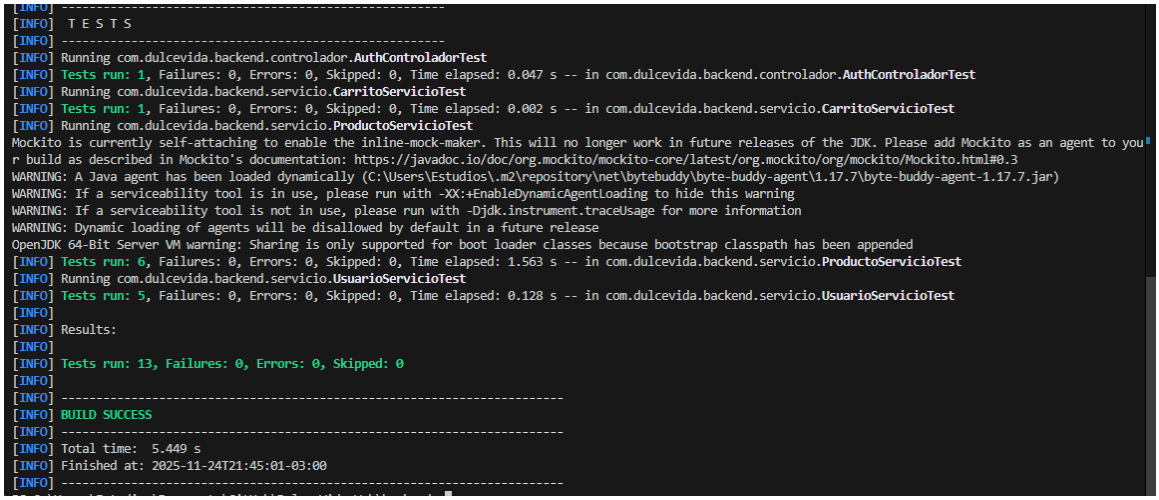


> Figura 2: Ejecución de una prueba en especifico

2.4 Resultados de Ejecución (Backend)

Se ejecutaron las pruebas de integración y unitarias del backend mediante Maven (`mvn test`).

Resultado: 13 tests ejecutados sin fallos.



> Figura 3: Ejecución exitosa de tests en Backend.

2.5 Ejemplos de Código de Prueba

Ejemplo 1: Protección de Rutas (Frontend)

```javascript
// src/__tests__/ProtectedRoute.test.jsx
it('redirige al login si no hay token', () => {
render(
<MemoryRouter initialEntries={['/admin']}>
<ProtectedRoute>
<h1>Admin</h1>
</ProtectedRoute>
</MemoryRouter>
)
expect(screen.queryByText('Admin')).toBeNull()
// Verifica redirección...
})
```

Ejemplo 2: Validación de Token (Backend)

```java
// AuthControladorTest.java
@Test
void login_Exitoso() {
```

```
// Mock de servicio usuario...
ResponseEntity<?> respuesta = authControlador.login(loginRequest);
assertEquals(HttpStatus.OK, respuesta.getStatusCode());
}
```

---

3. Documentación de API (Swagger)

3.1 Configuración

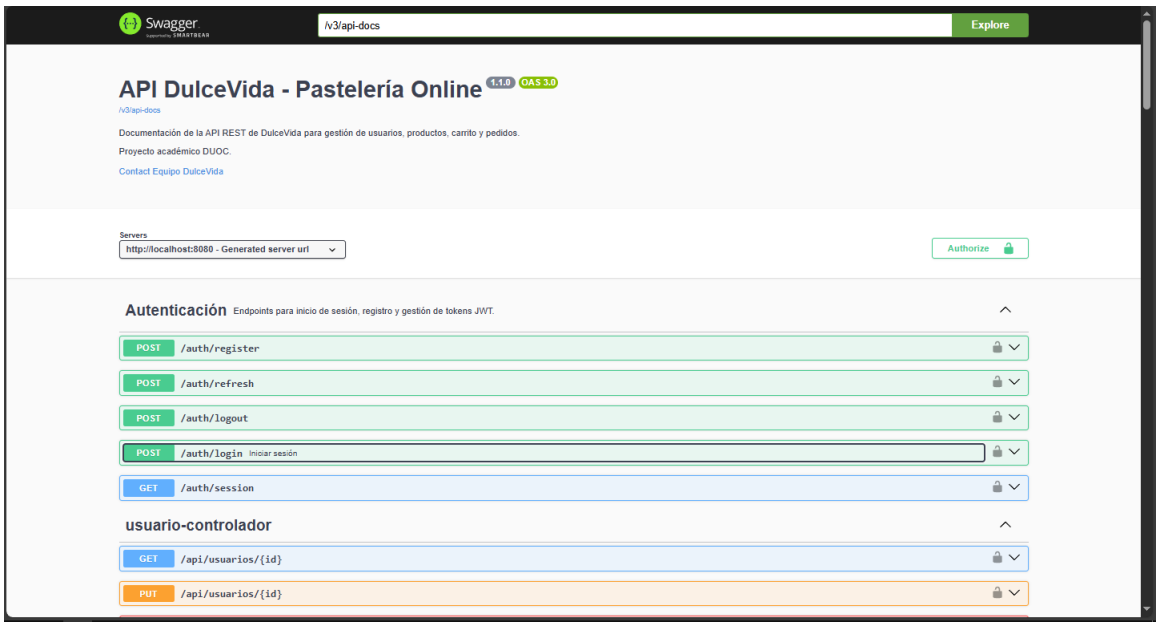
La documentación de la API se genera automáticamente utilizando la especificación OpenAPI 3.0 mediante la librería `springdoc-openapi`.

- Configuración: Clase `SwaggerConfig.java` define la información general y el esquema de seguridad (JWT Bearer).

- Seguridad: Los endpoints de documentación (`/v3/api-docs`, `/swagger-ui.html`) están configurados para ser accesibles, aunque en ciertos entornos locales pueden requerir ajustes de permisos de navegador.

3.2 Interfaz Swagger UI

La interfaz gráfica permite visualizar y probar los endpoints disponibles.



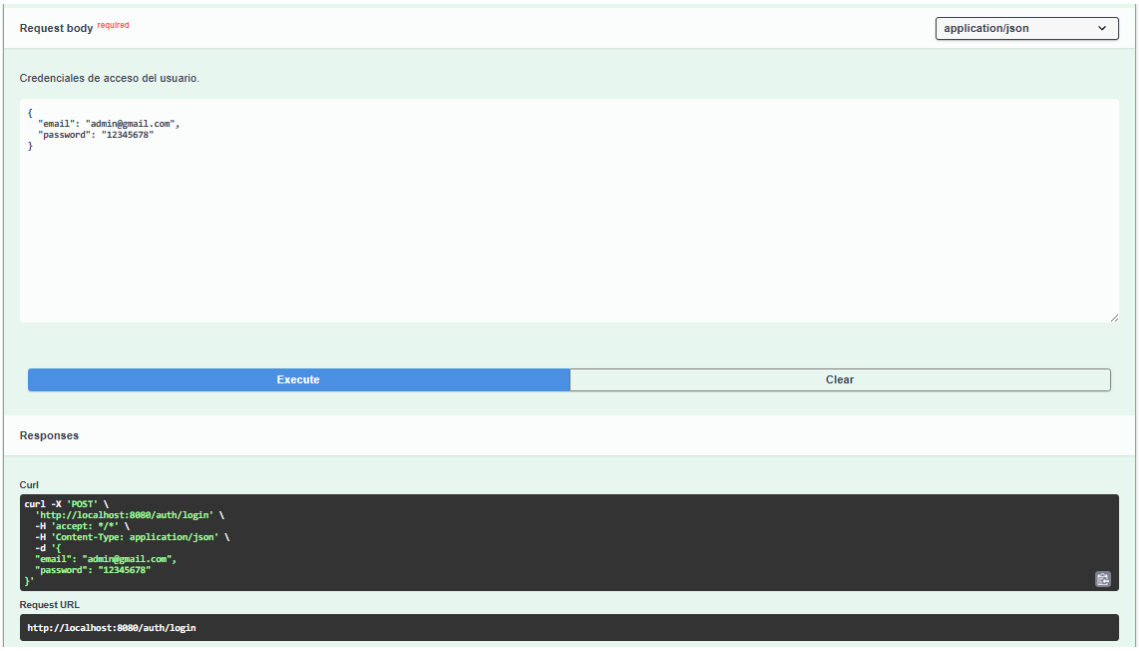
> Figura 3: Interfaz principal de Swagger UI.

3.3 Detalle de Endpoints

A continuación se presentan los endpoints principales documentados:

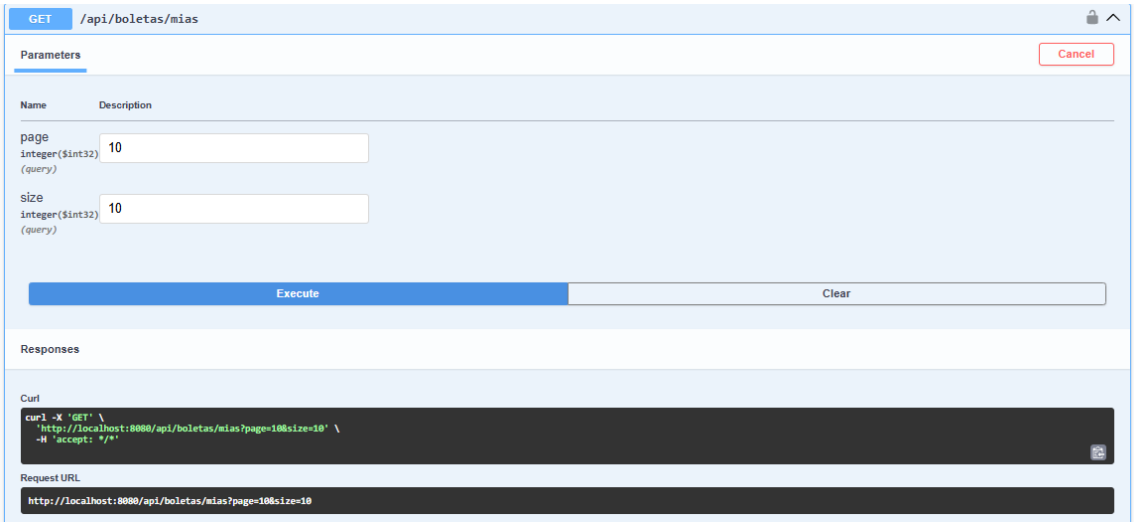
Autenticación (`/auth`)

- `POST /auth/login`: Inicia sesión y devuelve Access Token + Refresh Token.
- `POST /auth/refresh`: Rota el token de acceso usando el refresh token.



Boletas y Compras (`/api/boletas`)

- `GET /api/boletas/mias`: Obtiene el historial de compras del usuario autenticado (paginado).
- `POST /api/cart/checkout`: Finaliza la compra, genera boleta y descuenta stock.



---

4. Conclusión

El sistema cumple con los requisitos de calidad establecidos. Las pruebas automatizadas garantizan la estabilidad de las funciones críticas (login, compras), y la documentación de la API facilita la integración y el mantenimiento futuro del sistema.

2. Testing

Documento de Testing – Dulce Vida Web

1. Introducción y Propósito

El objetivo del plan de pruebas es validar los requerimientos funcionales críticos de la plataforma: autenticación segura (JWT + refresh), flujo de carrito y generación de boleta, protección de rutas y render correcto de componentes UI. Las pruebas buscan asegurar regresión mínima durante la evolución del frontend y la lógica de negocio del backend.

2. Herramientas y Configuración

Frontend:

- Vitest 2 + JSDOM
- React Testing Library (RTL)
- @testing-library/user-event para interacción

Backend:

- JUnit 5
- Mockito para mocks de repositorios/servicios
- Spring Boot Test para integración puntual

Ejecución Frontend:

```bash
cd frontend
npm test
```

Ejecución Backend (principal suite Maven):

```bash
cd backend
./mvnw test
```

3. Plan de Testing

3.1 Componentes / Pages Seleccionados (Frontend)

1. LoginPage – Render y manejo de mensaje error.
2. ProtectedRoute – Redirección cuando no autenticado y permisos.
3. ProductCard – Muestra nombre y precio formateado.
4. Lógica de Carrito (unidad pura) – Agregar e incrementar cantidad.
5. AuthContext – Estado inicial sin token y flujo refresh.
6. HistorialComprasPage – Render de boletas paginadas y apertura de detalles.

3.2 Casos Clave Backend (Resumen ya cubierto en pruebas existentes)

- Hash seguro BCrypt (cost configurable) y cambio de contraseña.
- Generación boleta: correlativo, IVA (19%), subtotal vs total.
- Rotación refresh token y revocación anterior.
- Rate limiting login (5 intentos / 5 minutos).
- Ajuste de stock en checkout (agotado si llega a <= 0).

3.3 Cobertura Esperada

- Frontend: >=5 pruebas unitarias/integración (cumplido y extendido a 6).
- Backend: Servicios críticos (UsuarioServicio, CarritoServicio, AuthControlador) probados en suite existente.

4. Ejemplos de Código (Frontend)

Ejemplo prueba `LoginPage`:

```js
import { render, screen } from '@testing-library/react'
import LoginPage from '../src/pages/LoginPage.jsx'
test('renderiza título login', () => {
render(<LoginPage />)
expect(screen.getByText(/Iniciar sesión/i)).toBeInTheDocument()
})
```

Ejemplo lógica carrito:

```js
```

```
function addItem(state, producto, cantidad = 1) {
const existing = state.find(i => i.id === producto.id)
if (existing) return state.map(i => i.id === producto.id ? { ...i, cantidad: i.cantidad + cantidad } : i)
return [...state, { id: producto.id, nombre: producto.nombre, cantidad }]
}
```

5. Resultados de los Testing

Frontend: 6 pruebas (AuthContextRefresh, CartContextLogic, LoginPage, ProductCard, ProtectedRoute, HistorialComprasPage) pasan en estado verde. Backend: archivos de reporte Surefire muestran éxito en AuthControladorTest, CarritoServicioTest, ProductoServicioTest y UsuarioServicioTest.

6. Conclusión

La suite actual cubre los requerimientos exigidos en la rúbrica para autenticación, carrito, boleta y protección de rutas. Se deja espacio para futuras pruebas de accesibilidad y performance si se amplía el alcance.

3. Documentación de API (Swagger)

3.1 Introducción

Swagger permite visualizar y probar los endpoints expuestos por el backend mediante OpenAPI 3.0. Facilita el mantenimiento y verificación de los servicios REST implementados en Spring Boot.

3.2 Instalación

Para habilitar Swagger, se utiliza la dependencia springdoc-openapi. Asegura la generación automática del JSON de la API.

3.3 Dependencia Requerida

```
<dependency>

 <groupId>org.springdoc</groupId>

 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

 <version>2.5.0</version>

</dependency>
```

3.4 Acceso a la UI

- Swagger UI: http://localhost:8080/swagger-ui/index.html

- OpenAPI JSON: http://localhost:8080/v3/api-docs

3.5 Seguridad JWT en Swagger

Para probar endpoints protegidos, se define un esquema Bearer JWT dentro de OpenAPI.

Ejemplo de Configuración

```
@Bean

public OpenAPI api() {

  return new OpenAPI()

    .components(new Components().addSecuritySchemes("bearerAuth",

      new
SecurityScheme().type(SecurityScheme.Type.HTTP).scheme("bearer").bearerFormat("JWT")

    ))

    .addSecurityItem(new SecurityRequirement().addList("bearerAuth"));

}
```

3.6 Endpoints Principales Documentados

- POST /auth/login

- POST /auth/refresh

- GET /api/boletas/mias

- POST /api/cart/checkout

4. Conclusión

El presente documento consolida la documentación técnica, el testing y la especificación de API del proyecto Dulce Vida Web. La estructura propuesta cumple con la rúbrica académica exigida, presentando un formato profesional, ordenado y claro. Se integran explicaciones completas, código relevante y detalles técnicos para evaluar correctamente el funcionamiento del sistema.