

The background is a dark blue field filled with various light blue and white icons. These include speech bubbles, a camera, a megaphone, a hand holding a clock, a smartphone, a tablet, a gear, and a hand holding a pen. A large, light blue rounded rectangle is centered on the page, serving as a backdrop for the main text.

FULLSTACK

RUSHAV

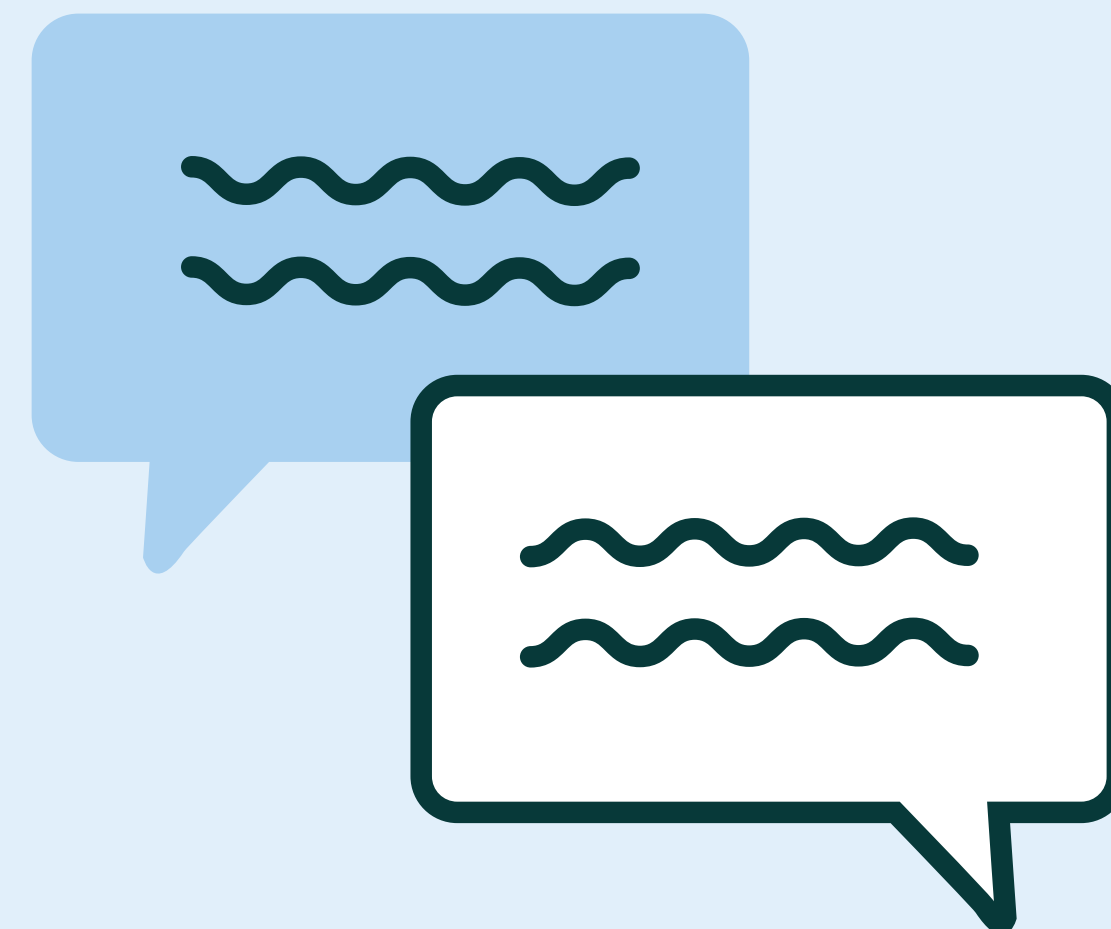
PROYECTO FULLSTACK 2



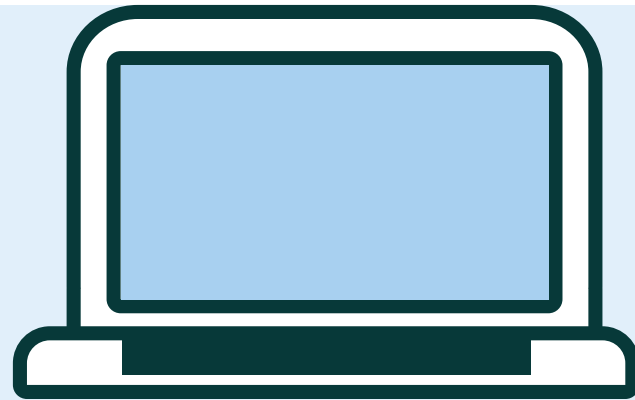
QUE HICIMOS?

Este proyecto se centró en la profesionalización de la aplicación inicial, llevando nuestro stack a un nivel de producción.

- Frontend Moderno: Migramos la interfaz de HTML a React para lograr un diseño más modular y reactivo.
- Full CRUD: Implementamos métodos CRUD completos en el Backend para gestionar los datos desde el panel de administrador.
- Persistencia Real: Se reemplazó el localStorage por una Base de Datos MySQL dedicada, asegurando la integridad y persistencia de la información.

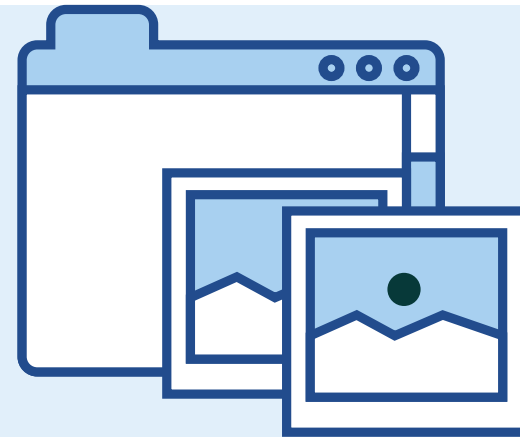


ESTRUCTURA



REACT FRONTEND

Para el desarrollo del Frontend utilizamos React, lo que nos permitió migrar de HTML a una arquitectura basada en componentes, volviendo la interfaz más modular y dinámica. Complementamos esto con Bootstrap para asegurar un diseño responsive y acelerar la maquetación del Panel de Administración



BACKEND

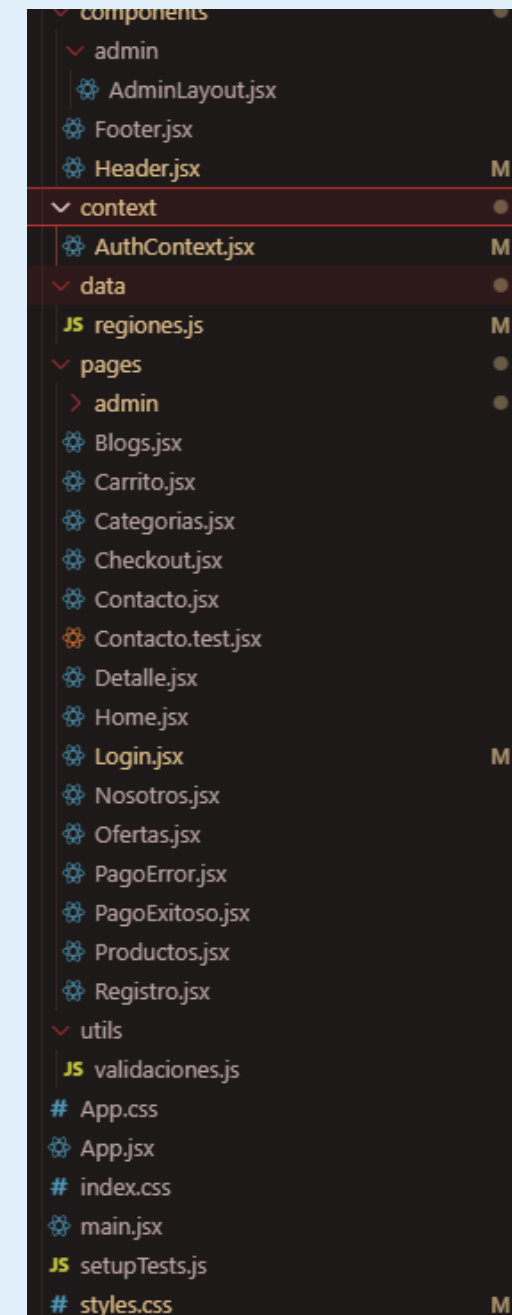
El Backend se construyó con Spring Boot para crear la API REST. Implementamos seguridad basada en JSON Web Tokens (JWT) para proteger el Panel de Administración y usamos Swagger para la documentación y prueba de todos nuestros endpoints.

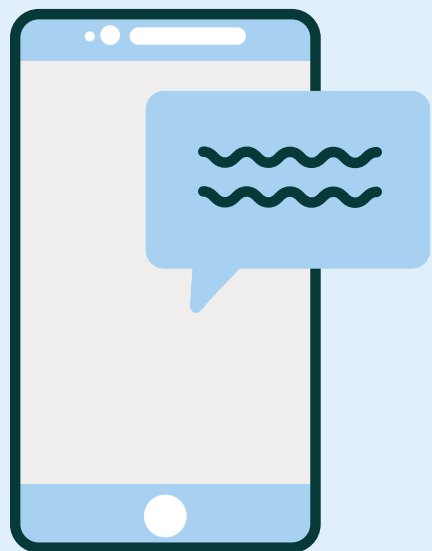


BASE DE DATOS

Para la persistencia de datos, implementamos una base de datos MySQL. Esto reemplazó el localStorage de la versión anterior, asegurando que todos los datos del administrador y del sistema se almacenen de forma persistente, estructurada y segura."

ESTRUCUTRA DEL PROYECTO



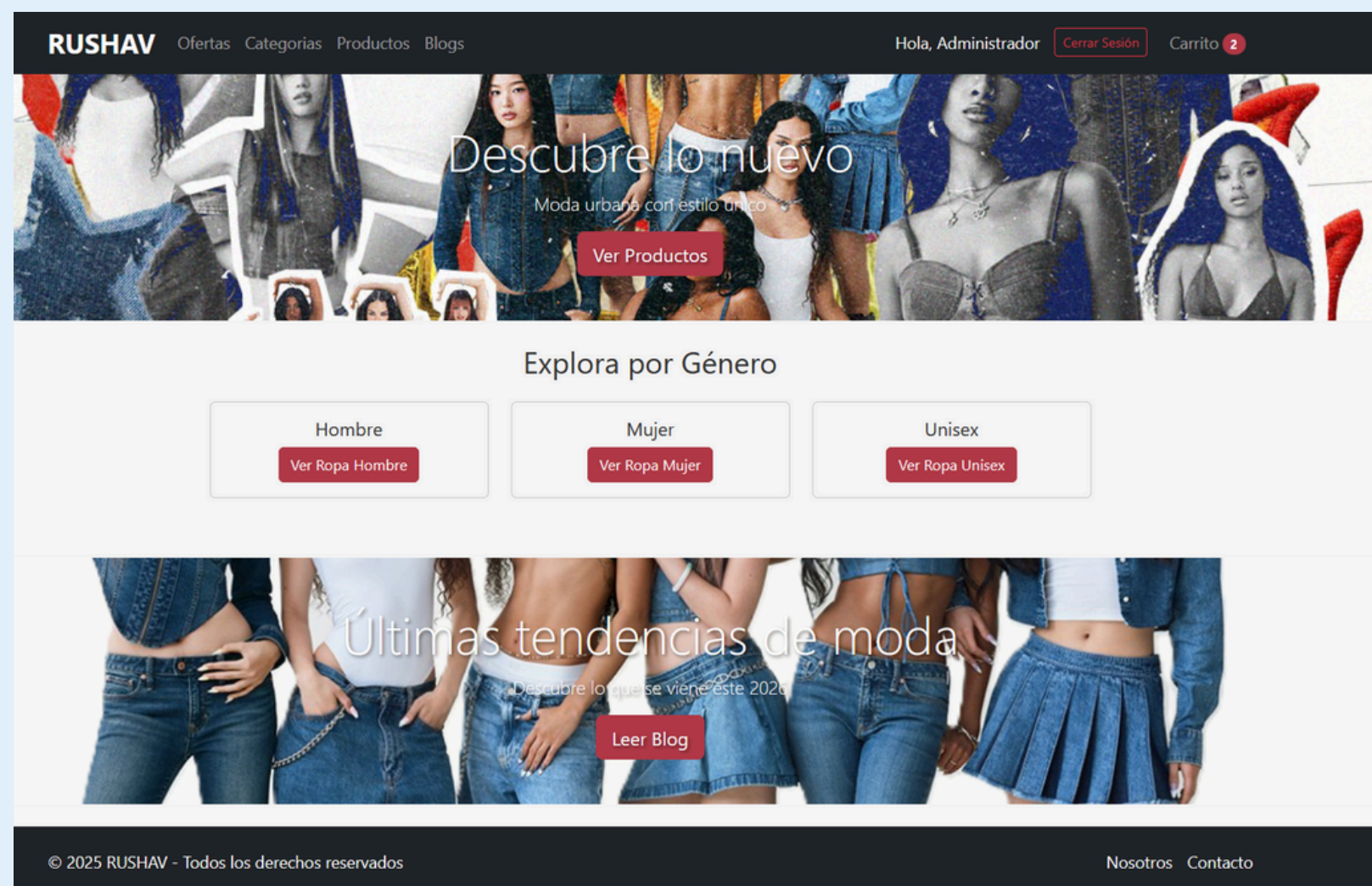


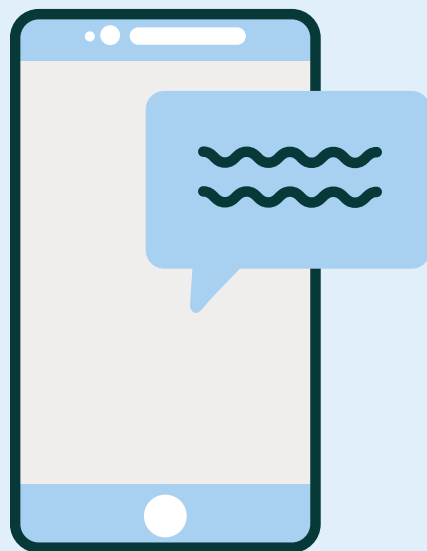
FRONTEND

Frontend con React: Interfaz de Usuario Dinámica

Para el desarrollo de la interfaz de usuario (UI), utilizamos React, lo que nos permitió migrar de un diseño estático en HTML a una arquitectura basada en componentes. Esto se traduce en una UI más modular, reutilizable y fácil de mantener.

Como se puede observar en la captura, la página principal está construida con componentes dinámicos que facilitan la navegación y la interacción del usuario. La implementación con React fue crucial para manejar eficientemente las actualizaciones de contenido, como la carga de productos o categorías, y para construir el panel de administración con funcionalidades CRUD de manera robusta.

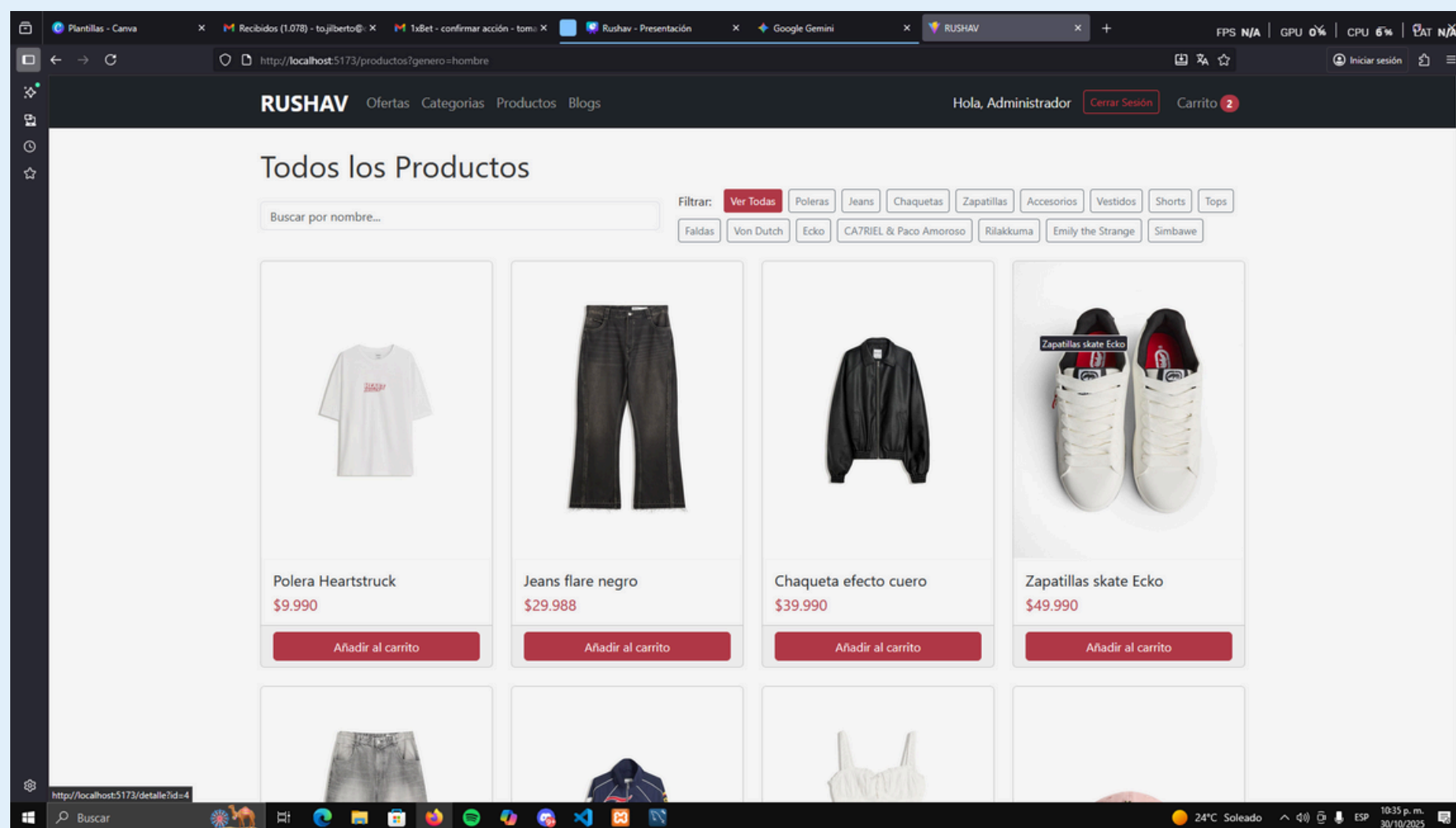


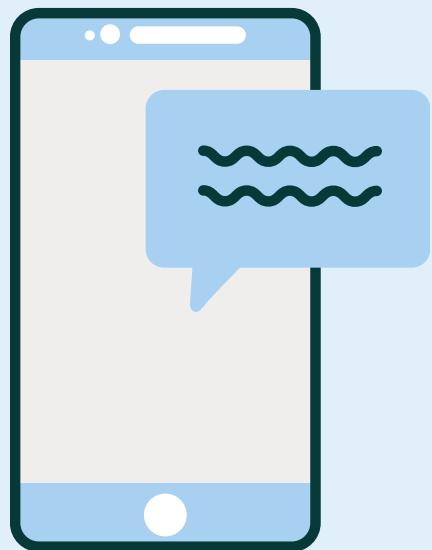


CATEGORIAS

La aplicación gestiona las categorías de productos (ej. Poleras, Jeans, Chaquetas) directamente desde la base de datos MySQL a través de la API REST.

- Funcionalidad Dinámica: React consume el listado de categorías y genera los botones de filtro de forma dinámica.
- Experiencia de Usuario: Cuando el usuario selecciona una categoría, React envía una petición al Backend, y la lista de productos se actualiza instantáneamente sin recargar la página. Esto demuestra una integración fluida entre el Frontend y el Backend.





CARRITO

Esta interfaz representa la etapa final del proceso de compra.

1. Validación y Totalización: El Frontend (React) presenta el resumen del carrito, calcula el Total y prepara los datos para la transacción.
2. Captura de Datos: El formulario asegura la captura de información crítica del cliente y la dirección de envío (Nombre, Apellidos, Correo, Región/Comuna).
3. Persistencia: Al hacer clic en "Pagar ahora", todos estos datos (artículos, total, información del cliente y dirección) son enviados a nuestra API REST de Spring Boot y se persisten en la base de datos MySQL como un nuevo registro de orden de compra o transacción, completando el ciclo de negocio.

[Inicio](#) [Categorias](#) [Productos](#) [Blogs](#) Hola, Rocío [Cerrar Sesión](#)

Finalizar Compra

Resumen del Carrito

Jeans flare negro x 1	\$29.990
Total	\$29.990

Dirección de entrega (*)

Calle y Número

Departamento (opcional)

Región

Metropolitana

Comuna

Talagante

Información del cliente (*)

Nombre

Rocío

Apellidos

Millaray Trujillo Plaza

Correo

rocio12@gmail.com

Pagar ahora \$29.990

[Inicio](#) [Categorias](#) [Productos](#) [Blogs](#)

X No se pudo realizar el pago

Hubo un error al procesar tu pago. Por favor, inténtalo de nuevo.

[VOLVER A REALIZAR EL PAGO](#) [Volver a productos](#)

Reservados todos los derechos reservados

✓ Pago Exitoso

¡Gracias por tu compra, Rocío!

Se ha realizado la compra con nro. #15758

Resumen de la Compra

Items:

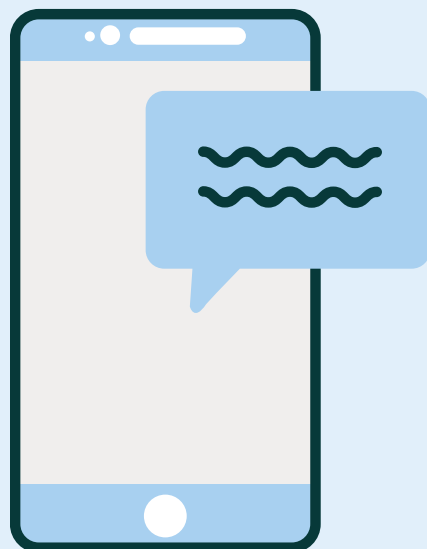
Jeans flare negro x 1	\$29.990
-----------------------	----------

Total Pagado: \$29.990

Dirección de Envío:

Avenida Siempreviva, 123, Talagante, Metropolitana

[Seguir comprando](#)







VISTA EXTRA OFERTAS

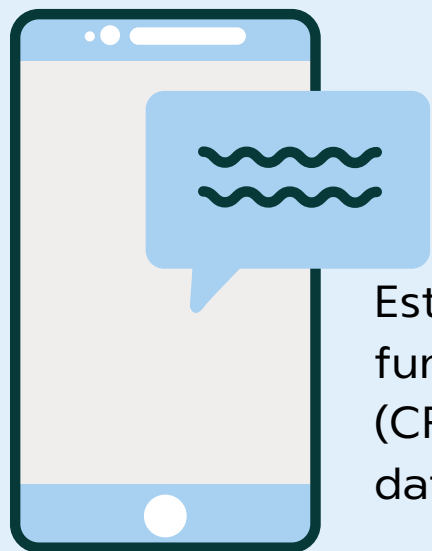
Esta vista demuestra el filtrado por lógica de negocio gestionado completamente por el Backend (Spring Boot).

1. Lógica del Backend: La API REST consulta la base de datos MySQL y aplica el filtro, entregando a React solo los productos que cumplen con la condición de oferta (en este caso, "Menos de \$17.000").
2. Dinamismo: Esto asegura que la información de ofertas sea siempre actual y evita que la lógica de precios se gestione en el Frontend, manteniendo la integridad de los datos en la capa del servidor.

RUSHAV [Ofertas](#) [Categorías](#) [Productos](#) [Blogs](#) Hola, Administrador [Cerrar Sesión](#) Carrito 2

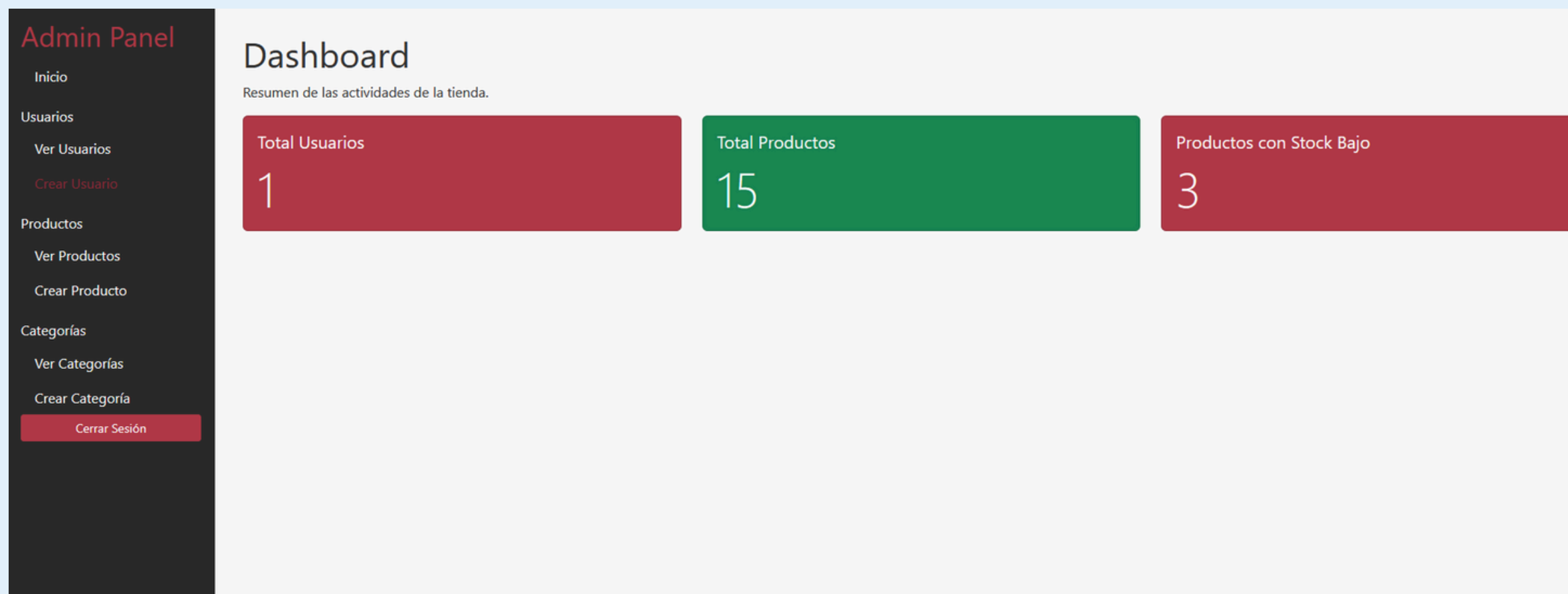
Productos en Oferta!!! (Menos de \$17.000)

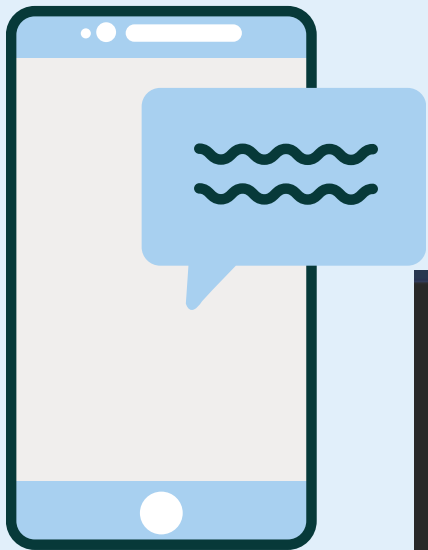
			
Polera Heartstruck \$9.990	Cortavientos azul gráfico oversize \$15.990	Gorro Von Dutch \$11.990	Top Von Dutch \$16.990
Añadir al carrito	Añadir al carrito	Añadir al carrito	Añadir al carrito



PANEL DE ADMINISTRACION

Este panel es el centro de control del sistema y fue implementado para la gestión de datos. Aquí se centralizan las operaciones fundamentales de la aplicación, soportadas por la API REST de Spring Boot. El panel permite la creación, lectura, actualización y eliminación (CRUD) de [menciona un recurso clave, ej: productos, usuarios o categorías], asegurando que la interfaz refleje los cambios en la base de datos MySQL en tiempo real.





PANEL DE ADMINISTRACION

Admin Panel

Inicio

Usuarios

Ver Usuarios

Crear Usuario

Productos

Ver Productos

Crear Producto

Categorías

Ver Categorías

Crear Categoría

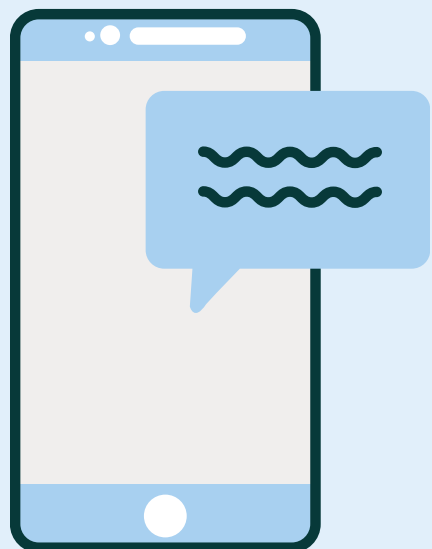
Cerrar Sesión

Gestión de Productos

Buscar por nombre...

-- Todas las Categorías --

ID	Nombre	Precio	Stock	Género	Activo	Acciones
1	Polera Heartstruck	\$9.990	20	hombre	Sí	<div>EditarEliminar</div>
2	Jeans flare negro	\$29.990	12	hombre	Sí	<div>EditarEliminar</div>
3	Chaqueta efecto cuero	\$39.990	8	hombre	Sí	<div>EditarEliminar</div>
4	Zapatillas skate Ecko	\$49.990	5	hombre	Sí	<div>EditarEliminar</div>
5	Jeans baggy gris	\$29.990	15	hombre	Sí	<div>EditarEliminar</div>
6	Cortavientos azul gráfico oversize	\$15.990	10	unisex	Sí	<div>EditarEliminar</div>
7	Vestido blanco bordado	\$25.990	13	mujer	Sí	<div>EditarEliminar</div>
8	Gorro Von Dutch	\$11.990	10	unisex	Sí	<div>EditarEliminar</div>
9	Chaqueta de mezdilla Ecko	\$19.990	10	hombre	Sí	<div>EditarEliminar</div>
10	Top Von Dutch	\$16.990	10	mujer	Sí	<div>EditarEliminar</div>
11	Falda denim Von Dutch	\$18.990	10	mujer	Sí	<div>EditarEliminar</div>
12	Short mini CA7RIEL & Paco Amoroso	\$25.990	4	unisex	Sí	<div>EditarEliminar</div>
13	Peludo Rilakkuma	\$35.990	7	mujer	Sí	<div>EditarEliminar</div>
14	Cartera Emily the Strange	\$17.990	2	mujer	Sí	<div>EditarEliminar</div>
15	Jorts parches estrellas	\$19.990	3	hombre	Sí	<div>EditarEliminar</div>



PANEL DE ADMINISTRACION

Editar Producto

Nombre (*)

Polera Heartstruck

Género

Hombre

Descripción (*)

Polera oversize con estampado exclusivo.

Precio (*)

5566

Stock (*)

5

Subir Imágenes (*)

Examinar... No se ha seleccionado

Imágenes Actuales

Imagen producto Imagen producto

Categorías (*)

☒ Poleras ☐ Jeans ☐ Chaquetas ☐ Zapatillas ☐ Accesorios ☐ Vestidos ☐ Shorts ☐ Tops ☐ Faldas ☐ Von Dutch ☐ Ecko

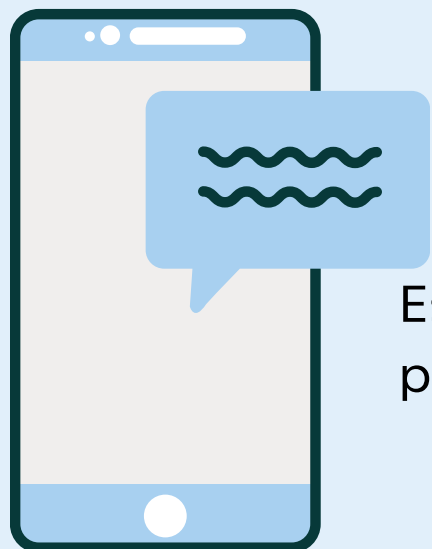
☐ CA7RIEL & Paco Amoroso ☐ Rilakkuma ☐ Emily the Strange ☐ Simbawe

☒ Producto Activo

localhost:5173

Producto actualizado con éxito

Aceptar



CRUD PARA PANEL

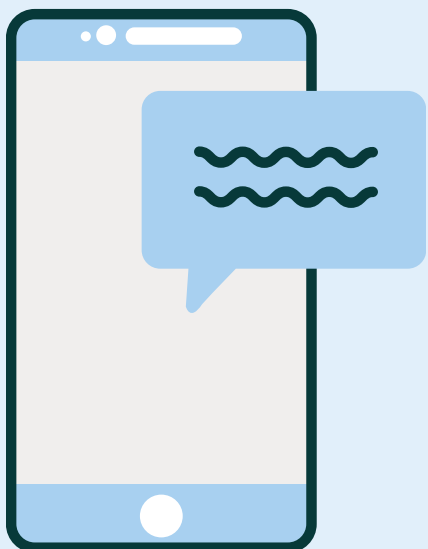
Este código muestra nuestro Controlador REST de Spring Boot, el cual gestiona la funcionalidad CRUD de los productos.

1. Mapeo REST: Utilizamos `@PostMapping`, `@PutMapping` y `@DeleteMapping` para crear, actualizar y eliminar productos, siguiendo los estándares de una API RESTful.
2. Documentación: Las anotaciones `@Operation` permiten que la API se auto-documente en Swagger UI.
3. Manejo de Respuestas: El uso de `ResponseEntity` nos permite devolver códigos de estado HTTP precisos (como 200, 204 o 404), haciendo nuestra API robusta y fácil de consumir por el Frontend de React.

```
@Operation(summary = "Crear nuevo producto", description = "Agrega un nuevo producto al catálogo")
@PostMapping
public ResponseEntity<Producto> crear(@Valid @RequestBody Producto p) {
    Producto creado = service.crear(p);
    return ResponseEntity.ok(creado);
}

@Operation(summary = "Actualizar producto", description = "Modifica los datos de un producto existente")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Producto actualizado exitosamente"),
    @ApiResponse(responseCode = "404", description = "Producto no encontrado")
})
@PutMapping("/{id}")
public ResponseEntity<Producto> actualizar(
    @Parameter(description = "ID del producto") @PathVariable Long id,
    @Valid @RequestBody Producto p) {
    try {
        Producto actualizado = service.actualizar(id, p);
        return ResponseEntity.ok(actualizado);
    } catch (RuntimeException ex) {
        return ResponseEntity.notFound().build();
    }
}

@Operation(summary = "Eliminar producto", description = "Elimina permanentemente un producto del sistema")
@DeleteMapping("/{id}")
public ResponseEntity<Void> eliminar(
    @Parameter(description = "ID del producto") @PathVariable Long id) {
    service.eliminar(id);
    return ResponseEntity.noContent().build();
}
```

SECURITYCONFIG

Este archivo es el guardia de seguridad de nuestro backend. Su trabajo es definir qué rutas son públicas y cuáles están protegidas."

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http

        .csrf(csrf -> csrf.disable()) // Deshabilita CSRF
        .cors(withDefaults()) // Habilita CORS usando tu Bean 'corsConfigurationSource'

        .authorizeHttpRequests(authz -> authz

            // --- 1. RUTAS PÚBLICAS (Cliente y Swagger) ---
            // Estas rutas son visibles para CUALQUIERA, sin autenticación.
            .requestMatchers(
                ...patterns:"/", "/index.html", "/static/**", "/favicon.ico",
                "/swagger-ui.html", "/swagger-ui/**", "/v3/api-docs/**",
                "/api-docs/**", "/swagger-config/**", "/swagger-resources/**", "/webjars/**"
            ).permitAll()

            // Endpoints de API públicos (Login, Registro, y ver productos/categorías)
            .requestMatchers(HttpMethod.POST, ...patterns:"/api/auth/login").permitAll()
            .requestMatchers(HttpMethod.POST, ...patterns:"/api/usuarios").permitAll() // Registro de clientes
            .requestMatchers(HttpMethod.GET, ...patterns:"/api/productos/**").permitAll() // <-- ARREGLO PARA RUTAS PÚBLICAS
            .requestMatchers(HttpMethod.GET, ...patterns:"/api/categorias/**").permitAll() // <-- ARREGLO PARA RUTAS PÚBLICAS
            .requestMatchers(HttpMethod.GET, ...patterns:"/api/archivos/descargar/**").permitAll()

            // --- 2. RUTAS PROTEGIDAS (Admin) ---
            // Todas las demás rutas que no sean las de arriba, requerirán autenticación.
            // Spring Security usará tu UserDetailsService (UsuarioServiceImpl) para validarlas.
            .requestMatchers(...patterns:"/api/dashboard/**").hasAnyAuthority(...authorities:"admin", "super-admin")
            .requestMatchers(HttpMethod.GET, ...patterns:"/api/usuarios/**").hasAnyAuthority(...authorities:"admin", "super-admin")
            .requestMatchers(HttpMethod.PUT, ...patterns:"/api/usuarios/**").hasAnyAuthority(...authorities:"admin", "super-admin")
            .requestMatchers(HttpMethod.DELETE, ...patterns:"/api/usuarios/**").hasAnyAuthority(...authorities:"admin", "super-admin")

            .requestMatchers(HttpMethod.POST, ...patterns:"/api/productos").hasAnyAuthority(...authorities:"admin", "super-admin")
            .requestMatchers(HttpMethod.PUT, ...patterns:"/api/productos/**").hasAnyAuthority(...authorities:"admin", "super-admin")
            .requestMatchers(HttpMethod.DELETE, ...patterns:"/api/productos/**").hasAnyAuthority(...authorities:"admin", "super-admin")

            .requestMatchers(HttpMethod.POST, ...patterns:"/api/categorias").hasAnyAuthority(...authorities:"admin", "super-admin")
```



AUTHCONTEXT

Este archivo es el cerebro de la seguridad en nuestro frontend. Es un Contexto de React que envuelve a toda la aplicación y se encarga de tres cosas

```
// (Función login)
const login = async (email, password) => {
  try {
    const response = await fetch(`${API_URL}/api/auth/login`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email: email, password: password }),
    });
    const data = await response.json();
    if (!response.ok) {
      throw new Error(data.mensaje || "Credenciales incorrectas");
    }
    if (data && data.usuarios && data.usuarios.id) {
      const basicToken = "Basic " + btoa(`${email}:${password}`);
      localStorage.setItem(USER_KEY, JSON.stringify(data.usuarios));
      localStorage.setItem(AUTH_TOKEN_KEY, basicToken);
      setCurrentUser(data.usuarios);
      setAuthToken(basicToken);
      return data.usuarios;
    } else {
      throw new Error("Respuesta de login inválida.");
    }
  } catch (error) {
    logout();
    throw error;
  }
};
```

```
// --- fetchProtegido (para JSON)
const fetchProtegido = useCallback(async (url, options = {}) => {
  const token = localStorage.getItem(AUTH_TOKEN_KEY);
  if (!token) {
    logout();
    throw new Error("Usuario no autenticado.");
  }
  const headers = {
    ...options.headers,
    "Authorization": token,
    "Content-Type": "application/json"
  };
  const response = await fetch(`${API_URL}${url}`, { ...options, headers });

  if (!response.ok) {
    const errorBody = await response.text(); // Lee el error como texto
    // Esto mostrará "El email ya está registrado"
    throw new Error(errorBody || `Error ${response.status}`);
  }

  if (response.status === 204) { // No Content (para DELETE)
    return { ok: true };
  }

  // Si todo está OK, leemos el JSON
  return response.json();
}, [logout]);
```

TESTS CON VITEST EN FRONTEND

```
PS D:\descargas\TiendaRushav\TiendaRushav> npm test
```

```
DEV v4.0.3 D:/descargas/TiendaRushav/TiendaRushav
```

- ✓ src/pages/Contacto.test.jsx (5 tests) 2506ms
 - ✓ Página de Contacto (5)
 - ✓ debe renderizar el formulario de contacto 216ms
 - ✓ debe mostrar errores de validación si el formulario está vacío 119ms
 - ✓ debe mostrar un error de correo inválido 993ms
 - ✓ debe actualizar el estado y el contador al escribir en el mensaje 211ms
 - ✓ debe mostrar mensaje de éxito al enviar un formulario válido 963ms

Test Files 1 passed (1)

Tests 5 passed (5)

Start at 22:21:08

Duration 6.50s (transform 226ms, setup 620ms, collect 322ms, tests 2.51s, en

PASS Waiting for file changes...

press h to show help, press q to quit

```
it('debe renderizar el formulario de contacto', () => {
  render(<Contacto />);

  // Verificamos que el título y los campos existan
  expect(screen.getByRole('heading', { name: /Contacto/i })).toBeInTheDocument();
  expect(screen.getByLabelText(/Nombre completo */i)).toBeInTheDocument();
  expect(screen.getByLabelText(/Correo electrónico */i)).toBeInTheDocument();
  expect(screen.getByLabelText(/Mensaje */i)).toBeInTheDocument();
});

it('debe mostrar errores de validación si el formulario está vacío', async () => {
  render(<Contacto />);
  const user = userEvent.setup();

  // Simula un clic en el botón de enviar
  await user.click(screen.getByRole('button', { name: /Enviar Mensaje/i }));

  // Busca los mensajes de error
  expect(await screen.findByText(/El nombre es obligatorio/i)).toBeInTheDocument();
  expect(screen.getByText(/El correo es obligatorio/i)).toBeInTheDocument();
  expect(screen.getByText(/El mensaje es obligatorio/i)).toBeInTheDocument();
});

it('debe mostrar un error de correo inválido', async () => {
  render(<Contacto />);
  const user = userEvent.setup();


  await user.type(screen.getByLabelText(/Nombre completo */i), 'Tester');
  await user.type(screen.getByLabelText(/Correo electrónico */i), 'correo@invalido.com');
  await user.type(screen.getByLabelText(/Mensaje */i), 'Un mensaje');


  await user.click(screen.getByRole('button', { name: /Enviar Mensaje/i }));















  // Busca el error específico del correo
  expect(await screen.findByText(/El correo debe ser @dos.cl/i)).toBeInTheDocument();
});
```

TESTS CON MOCKITO EN BACKEND

TESTING

Filter (e.g. text, !exclude, @tag) 

14/14 16.8s 

- ✓  rushav-backend 3.0s
 - ✓  {} com.rushav.rushav_backend 23ms
 - > ✓  RushavBackendApplicationTests 23ms
 - ✓  {} com.rushav.rushav_backend.controllers 114ms
 - > ✓  AuthControllerTest 114ms
 - ✓  {} com.rushav.rushav_backend.entities 9.0ms
 - > ✓  ProductoTest 9.0ms
 - ✓  {} com.rushav.rushav_backend.services.impl 2.9s
 - ✓  ProductoServiceImplTest 148ms
 - ✓  testProductosConStockBajo() 5.0ms
 - ✓  testContarTotalProductos() 143ms
 - ✓  UsuarioServiceImplTest 2.7s
 - ✓  testCrearUsuario() 2.7s
 - ✓  testContarTotalUsuarios() 10ms

```
@Test
public void testCrearUsuario() {
    // --- Preparación (Arrange) ---
    Usuario usuario = new Usuario();
    usuario.setNombre(nombre:"Test User");
    usuario.setEmail(email:"test@test.com");
    usuario.setPassword(password:"password123"); // Contraseña en texto plano

    Usuario usuarioGuardado = new Usuario();
    usuarioGuardado.setId(id:1L);
    usuarioGuardado.setNombre(nombre:"Test User");
    usuarioGuardado.setPassword(password:"hash_encriptado_simulado"); // Contraseña encriptada

    // --- 3. SIMULA EL COMPORTAMIENTO DEL ENCODER ---
    // Dile a Mockito: "Cuando se llame a encode() con 'password123',
    // devuelve este hash simulado"
    when(passwordEncoder.encode("password123")).thenReturn("hash_encriptado_simulado");

    // Simula el guardado en el repositorio
    when(usuarioRepository.save(any(Usuario.class))).thenReturn(usuarioGuardado);

    // --- Ejecución (Act) ---
    Usuario resultado = usuarioService.crear(usuario);

    // --- Verificación (Assert) ---
    assertNotNull(resultado.getId());
    assertEquals("Test User", resultado.getNombre());

    // Verifica que la contraseña en el resultado sea la encriptada
    assertEquals("hash_encriptado_simulado", resultado.getPassword());

    // Verifica que el método encode sí fue llamado
    verify(passwordEncoder, times(1)).encode("password123");
}
```

TRABAJO COLABORATIVO GITHUB

PARA EL TRABAJO EN EQUIPO, UTILIZAMOS GITHUB PARA EL CONTROL DE VERSIONES. ESTO FUE FUNDAMENTAL PARA LA COLABORACIÓN, YA QUE IMPLEMENTAMOS UNA ESTRATEGIA DE BRANCHES (RAMAS) SEPARADAS:

- UTILIZAMOS UN BRANCH ESPECÍFICO PARA EL FRONTEND (REACT) Y OTRO BRANCH ESPECÍFICO PARA EL BACKEND (SPRING BOOT).
- ESTO NOS PERMITIÓ TRABAJAR EN PARALELO, INTEGRAR LOS CAMBIOS DE MANERA ORDENADA Y EVITAR CONFLICTOS DE CÓDIGO ENTRE LAS CAPAS, ASEGURANDO LA ESTABILIDAD DEL PROYECTO EN TODO MOMENTO.

```
MINGW64/c/Users/Usuario/Desktop/frontend/TiendaRushav

Usuario@EVA-01 MINGW64 ~/Desktop/frontend/TiendaRushav (frontend)
$ git checkout frontend
M   index.html
M   src/components/Header.jsx
M   src/context/AuthContext.jsx
M   src/data/regiones.js
M   src/pages/Login.jsx
M   src/pages/Productos.jsx
M   src/pages/Registro.jsx
M   src/pages/admin/AdminCategoriasForm.jsx
M   src/pages/admin/AdminCategoriasListar.jsx
M   src/pages/admin/AdminDashboard.jsx
M   src/pages/admin/AdminProductosForm.jsx
M   src/pages/admin/AdminProductosListar.jsx
M   src/pages/admin/AdminUsuariosForm.jsx
M   src/pages/admin/AdminUsuariosListar.jsx
M   src/styles.css
Already on 'frontend'
Your branch is up to date with 'origin/frontend'.

Usuario@EVA-01 MINGW64 ~/Desktop/frontend/TiendaRushav (frontend)
$ git add .

Usuario@EVA-01 MINGW64 ~/Desktop/frontend/TiendaRushav (frontend)
$ git commit -m "Finalizando cambios del frontend (dashboard, validaciones header)"
[frontend 0cf9475] Finalizando cambios del frontend (dashboard, validaciones header)
 15 files changed, 768 insertions(+), 577 deletions(-)

Usuario@EVA-01 MINGW64 ~/Desktop/frontend/TiendaRushav (frontend)
$ git push origin frontend
Enumerating objects: 45, done.
Counting objects: 100% (45/45), done.
Delta compression using up to 8 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (23/23), 13.17 KiB | 1.46 MiB/s, done.
Total 23 (delta 16), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (16/16), completed with 16 local objects.
To https://github.com/rociomtrujillo/RUSHAV2
 2786365..0cf9475  frontend -> frontend

Usuario@EVA-01 MINGW64 ~/Desktop/frontend/TiendaRushav (frontend)
$ |
```

```
Usuario@EVA-01 MINGW64 ~/Desktop/backend (backend)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Usuario@EVA-01 MINGW64 ~/Desktop/backend (main)
$ git merge backend
Updating 8bf776e..31d3836
Fast-forward
 rushav-backend/pom.xml | 169 ++++++-----
.../rushav_backend/RushavBackendApplication.java | 1 -
.../rushav/rushav_backend/config/AppConfig.java | 15 ++
.../rushav_backend/config/OpenApiConfig.java | 24 +++
.../rushav_backend/config/SecurityConfig.java | 86 ++++++++
.../rushav_backend/controllers/AuthController.java | 23 +-
.../controllers/CategoriaController.java | 38 +---
.../controllers/DashboardController.java | 4 +
.../controllers/FileUploadController.java | 154 ++++++
.../controllers/ProductoController.java | 213 ++++++
.../controllers/UsuarioController.java | 33 +-
.../rushav_backend/dtos/FileUploadResponse.java | 31 +++
.../rushav/rushav_backend/dtos/LoginRequest.java | 7 +
.../rushav/rushav_backend/dtos/LoginResponse.java | 6 +
.../rushav/rushav_backend/entities/Categoria.java | 8 +
.../rushav/rushav_backend/entities/Producto.java | 14 +-
.../rushav/rushav_backend/entities/Usuario.java | 53 +++++
.../services/FileStorageService.java | 55 ++++++
.../services/impl/CategoriaServiceImpl.java | 97 ++++++
.../services/impl/UsuarioServiceImpl.java | 106 ++++++
.../src/main/resources/application.properties | 11 +-
rushav-backend/src/main/resources/data.sql | 24 +--
.../controllers/AuthControllerTest.java | 131 ++++++
23 files changed, 1091 insertions(+), 212 deletions(-)
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/config/AppConfig.java
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/config/OpenApiConfig.java
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/config/SecurityConfig.java
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/controllers/FileUploadController.java
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/dtos/FileUploadResponse.java
create mode 100644 rushav-backend/src/main/java/com/rushav/rushav_backend/services/FileStorageService.java

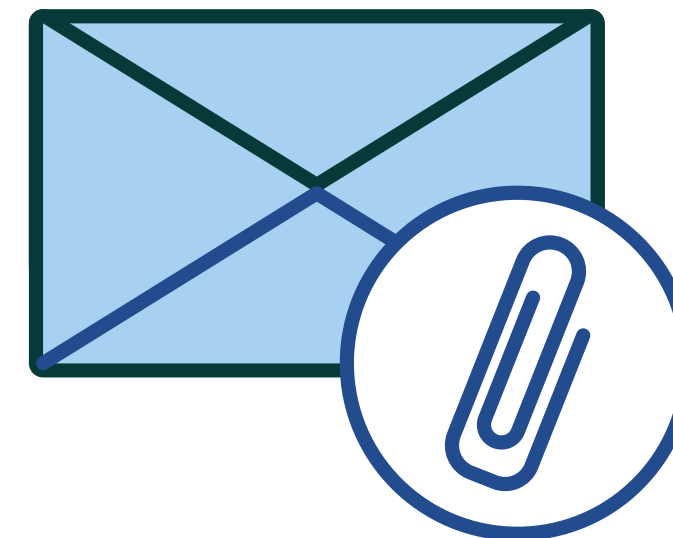
Usuario@EVA-01 MINGW64 ~/Desktop/backend (main)
$ git merge frontend
merge: frontend - not something we can merge

Usuario@EVA-01 MINGW64 ~/Desktop/backend (main)
$ AC

Usuario@EVA-01 MINGW64 ~/Desktop/backend (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/rociomtrujillo/RUSHAV2.git
 8bf776e..31d3836  main -> main
```




RESUMEN



01

Reconstruimos el Frontend estático (HTML) en React para establecer una arquitectura modular basada en componentes, mejorando la escalabilidad y la interactividad de la interfaz.

02

Utilizamos Bootstrap para la maquetación, lo que nos permitió acelerar el diseño y asegurar que el Panel de Administración sea completamente responsivo (funcional en cualquier dispositivo).

03

Desarrollamos el Backend utilizando el framework Spring Boot para exponer una API RESTful robusta, que sirve como el cerebro de la aplicación.

04

Reemplazamos la solución temporal de localStorage por una Base de Datos MySQL dedicada, logrando una persistencia de datos real y estructurada para todo el sistema.

05

Implementamos la seguridad mediante JSON Web Tokens (JWT) para proteger los endpoints y utilizamos el algoritmo BCrypt (con salt) para la encriptación segura de las contraseñas.

06

Desarrollamos la funcionalidad CRUD completa para la gestión de datos en el Backend y la documentamos automáticamente mediante Swagger UI (SpringDoc), facilitando las pruebas y el consumo de la API.

MUCHAS GRACIAS

POR SU ATENCIÓN