API ENDPOINTS COMPLETOS

Sistema de Inventario - Documentación de API REST

ÍNDICE DE ENDPOINTS

Categoría	Cantidad	Admin	Vendedor
Usuarios	18	☑ 18	☑ 12
Productos	8	☑ 8	☑ 8
Categorías	6	☑ 6	☑ 6
Proveedores	6	☑ 6	☑ 6
Ventas	7	☑ 7	☑ 7
Movimientos	5	☑ 5	☑ 5
Dashboard	8	☑ 8	☑ 6
TOTAL	**58**	**58**	**50**

AUTENTICACIÓN Y USUARIOS

1. POST /api/usuarios/login

Propósito: Iniciar sesión en el sistema

Rate Limit: 5 intentos cada 15 minutos

Request Body:

```
{
  "email": "admin@rectificadora.com",
  "password": "admin123"
}
```

Response Success (200):

```
{
  "message": "Login exitoso",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "usuario": {
    "idUsuario": 1,
    "nombre": "Administrador",
    "email": "admin@rectificadora.com",
```

```
    "idRol": 1,
    "nombreRol": "Administrador",
    "activo": true
  }
}
```

Response Error (401):

```
{
  "message": "Credenciales incorrectas",
  "intentos_restantes": 4
}
```

Response Error (429 - Rate Limit):

```
{
  "message": "Demasiados intentos de inicio de sesión. Intenta nuevamente en 15 minutos.",
  "retryAfter": 897
}
```

Código Backend (index.js líneas 200-280):

```
app.post('/api/usuarios/login', loginLimiter, [
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 6 })
], async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { email, password } = req.body;

  try {
    // Buscar usuario
    const [usuarios] = await pool.query(
      'SELECT u.*, r.nombre as nombreRol FROM Usuario u JOIN Rol r ON u.idRol = r.idRol WHERE u.email = ?',
      [email]
    );

    if (usuarios.length === 0) {
      return res.status(401).json({ message: 'Credenciales incorrectas' });
    }

    const usuario = usuarios[0];
```

```
    // Verificar si está activo
    if (!usuario.activo) {
      return res.status(403).json({ message: 'Usuario desactivado' });
    }

    // Comparar contraseña
    const passwordValida = await bcrypt.compare(password, usuario.contraseña);
    if (!passwordValida) {
      return res.status(401).json({ message: 'Credenciales incorrectas' });
    }

    // Registrar inicio de sesión
    await pool.query(
      'UPDATE Usuario SET fechaInicioSesion = NOW(), fechaFinSesion = NULL WHERE
idUsuario = ?',
      [usuario.idUsuario]
    );

    // Generar token JWT
    const token = jwt.sign(
      {
        idUsuario: usuario.idUsuario,
        email: usuario.email,
        idRol: usuario.idRol
      },
      JWT_SECRET,
      { expiresIn: '8h' }
    );

    res.json({
      message: 'Login exitoso',
      token,
      usuario: {
        idUsuario: usuario.idUsuario,
        nombre: usuario.nombre,
        email: usuario.email,
        idRol: usuario.idRol,
        nombreRol: usuario.nombreRol,
        activo: usuario.activo
      }
    });
  } catch (error) {
    console.error('Error en login:', error);
    res.status(500).json({ message: 'Error en el servidor' });
  }
});
```

---

2. POST /api/usuarios/registro

Propósito: Crear nuevo usuario (Solo Admin puede crear otros admins)

Requiere: Token JWT

Request Body:

```
{
  "nombre": "Juan Pérez",
  "email": "juan@rectificadora.com",
  "password": "password123",
  "idRol": 2
}
```

Validaciones:

- Email único en la base de datos
- Password mínimo 6 caracteres
- idRol debe ser 1 o 2
- Solo Admin puede crear idRol = 1

Response Success (201):

```
{
  "message": "Usuario registrado exitosamente",
  "idUsuario": 5
}
```

---

3. GET /api/usuarios/me

Propósito: Obtener datos del usuario actual

Requiere: Token JWT en header `Authorization: Bearer TOKEN`

Response (200):

```
{
  "usuario": {
    "idUsuario": 1,
    "nombre": "Administrador",
    "email": "admin@rectificadora.com",
    "telefono": "942123456",
    "direccion": "Tarapoto, San Martín",
    "idRol": 1,
    "nombreRol": "Administrador",
    "activo": true,
    "fechaCreacion": "2025-01-15T10:30:00.000Z",
    "fechaInicioSesion": "2025-01-21T14:20:00.000Z"
  }
}
```

Uso en Frontend:

```
const response = await apiFetch('/api/usuarios/me');
const data = await response.json();
setUserData(data.usuario);
```

---

4. PUT /api/usuarios/me

Propósito: Actualizar perfil del usuario actual

Requiere: Token JWT

Request Body:

```
{
  "nombre": "Juan Carlos Pérez",
  "telefono": "942555666",
  "direccion": "Av. Principal 123, Tarapoto"
}
```

Response (200):

```
{
  "message": "Perfil actualizado correctamente",
  "usuario": { ... }
}
```

---

5. POST /api/usuarios/cambiar-contraseña

Propósito: Cambiar contraseña del usuario actual

Requiere: Token JWT

Request Body:

```
{
  "currentPassword": "password_antigua",
  "newPassword": "password_nueva_segura"
}
```

Validaciones:

- currentPassword debe coincidir con la actual
- newPassword mínimo 6 caracteres
- newPassword != currentPassword

Response Success (200):

```
{
  "message": "Contraseña actualizada correctamente"
}
```

Response Error (401):

```
{
  "message": "La contraseña actual es incorrecta"
}
```

---

6. PATCH /api/usuarios/logout

Propósito: Registrar cierre de sesión

Requiere: Token JWT

Response (200):

```
{
  "message": "Sesión cerrada correctamente"
}
```

Acción en BD:

```
UPDATE Usuario
SET fechaFinSesion = NOW()
WHERE idUsuario = ?
```

---

7. POST /api/usuarios/forgot-password

Propósito: Solicitar recuperación de contraseña

No requiere token

Request Body:

```
{
  "email": "usuario@rectificadora.com"
}
```

Proceso:

1. Verifica que el email exista
2. Genera token aleatorio de 32 bytes
3. Guarda token y fecha de expiración (1 hora) en BD
4. Envía email con link de recuperación

Response (200):

```
{
  "message": "Se ha enviado un correo con instrucciones para restablecer tu
contraseña"
}
```

Email enviado:

```
Asunto: 🔐 Recuperación de Contraseña

Hola [Nombre],

Haz click aquí para restablecer tu contraseña:
http://localhost:5174/restablecer-contrasena/[TOKEN]

Este enlace expira en 1 hora.
```

---

8. POST /api/usuarios/reset-password/:token

Propósito: Restablecer contraseña con token

No requiere JWT (usa token de URL)

Request Body:

```
{
  "newPassword": "nueva_password_segura"
}
```

Validaciones:

1. Token debe existir en BD

2. Token no debe estar expirado
3. newPassword mínimo 6 caracteres

Response Success (200):

```
{
  "message": "Contraseña restablecida correctamente"
}
```

Response Error (400):

```
{
  "message": "Token inválido o expirado"
}
```

---

9. GET /api/usuarios (Solo Admin)

Propósito: Listar todos los usuarios

Requiere: Token JWT + idRol = 1

Response (200):

```
{
  "usuarios": [
    {
      "idUsuario": 1,
      "nombre": "Administrador",
      "email": "admin@rectificadora.com",
      "telefono": "942123456",
      "idRol": 1,
      "nombreRol": "Administrador",
      "activo": true,
      "fechaCreacion": "2025-01-15T10:30:00.000Z"
    },
    {
      "idUsuario": 2,
      "nombre": "Vendedor 1",
      "email": "vendedor1@rectificadora.com",
      "telefono": "942987654",
      "idRol": 2,
      "nombreRol": "Vendedor",
      "activo": true,
      "fechaCreacion": "2025-01-16T09:15:00.000Z"
    }
  ]
}
```

---

10. PUT /api/usuarios/:id (Solo Admin)

Propósito: Actualizar datos de cualquier usuario

Requiere: Token JWT + idRol = 1

Request Body:

```
{
  "nombre": "Nuevo Nombre",
  "email": "nuevo@email.com",
  "telefono": "999888777",
  "idRol": 2
}
```

---

11. PUT /api/usuarios/:id/estado (Solo Admin)

Propósito: Activar/Desactivar usuario

Requiere: Token JWT + idRol = 1

Request Body:

```
{
  "activo": false
}
```

Restricción: No se puede desactivar a sí mismo

---

12. POST /api/usuarios/refresh

Propósito: Renovar token JWT antes de que expire

Requiere: Token JWT válido

Response (200):

```
{
  "token": "nuevo_token_jwt_con_8_horas_mas"
}
```

Uso Frontend (api.js):

```
export async function refreshTokenIfNeeded() {
  const token = getToken();
  const payload = JSON.parse(atob(token.split('.')[1]));
  const now = Math.floor(Date.now() / 1000);
  const timeUntilExpiry = payload.exp - now;

  // Si falta menos de 30 minutos, renovar
  if (timeUntilExpiry < 1800) {
    const response = await apiFetch('/api/usuarios/refresh', {
      method: 'POST'
    });
    const data = await response.json();
    setToken(data.token);
  }
}
```

PRODUCTOS

1. GET /api/productos

Propósito: Listar todos los productos con sus relaciones

Requiere: Token JWT

Response (200):

```
{
  "productos": [
    {
      "idProducto": 1,
      "codigo": "REP-001",
      "nombre": "Pistón de Motor 125cc",
      "descripcion": "Pistón compatible con motores Bajaj",
      "precioUnitario": 85.50,
      "stock": 45,
      "stockMinimo": 10,
      "activo": true,
      "idCategoria": 1,
      "nombreCategoria": "Repuestos de Motor",
      "idProveedor": 2,
      "nombreProveedor": "Repuestos SAC",
      "fechaCreacion": "2025-01-10T08:00:00.000Z"
    }
  ]
}
```

Query SQL (index.js):

```sql
SELECT
  p.*,
  c.nombre as nombreCategoria,
  pr.nombre as nombreProveedor
FROM Producto p
LEFT JOIN Categoria c ON p.idCategoria = c.idCategoria
LEFT JOIN Proveedor pr ON p.idProveedor = pr.idProveedor
WHERE p.activo = 1
ORDER BY p.nombre
```

---

2. GET /api/productos/:id

Propósito: Obtener detalles de un producto específico

Response (200):

```
{
  "producto": {
    "idProducto": 1,
    "codigo": "REP-001",
    "nombre": "Pistón de Motor 125cc",
    "descripcion": "...",
    "precioUnitario": 85.50,
    "stock": 45,
    "stockMinimo": 10,
    "activo": true,
    "idCategoria": 1,
    "idProveedor": 2
  }
}
```

---

3. POST /api/productos

Propósito: Crear nuevo producto

Requiere: Token JWT

Request Body:

```
{
  "codigo": "REP-050",
  "nombre": "Filtro de Aceite",
  "descripcion": "Filtro compatible con motores 150cc-200cc",
  "precioUnitario": 25.00,
  "stock": 100,
  "stockMinimo": 20,
  "idCategoria": 3,
```

```
  "idProveedor": 5
}
```

Validaciones:

- codigo único
- precioUnitario > 0
- stock >= 0
- stockMinimo >= 0
- idCategoria debe existir
- idProveedor debe existir

Response Success (201):

```
{
  "message": "Producto creado exitosamente",
  "idProducto": 51
}
```

---

4. PUT /api/productos/:id

Propósito: Actualizar producto existente

Request Body: (igual que POST)

Response (200):

```
{
  "message": "Producto actualizado correctamente"
}
```

---

5. DELETE /api/productos/:id

Propósito: Eliminar producto (soft delete)

Acción: Cambia `activo = 0` en lugar de borrar

Response (200):

```
{
  "message": "Producto eliminado correctamente"
}
```

---

6. GET /api/productos/bajo-stock

Propósito: Listar productos con stock bajo el mínimo

Response (200):

```
{
  "productos": [
    {
      "idProducto": 15,
      "nombre": "Cadena de Transmisión",
      "stock": 5,
      "stockMinimo": 10,
      "diferencia": -5
    }
  ]
}
```

Query SQL:

```
SELECT * FROM Producto
WHERE stock < stockMinimo
AND activo = 1
```

---

7. GET /api/productos/buscar

Propósito: Buscar productos por nombre o código

Query Params: `?q=pistón`

Response (200):

```
{
  "productos": [
    { "idProducto": 1, "nombre": "Pistón de Motor 125cc", ... },
    { "idProducto": 12, "nombre": "Pistón de Motor 150cc", ... }
  ]
}
```

Query SQL:

```
SELECT * FROM Producto
WHERE (nombre LIKE '%pistón%' OR codigo LIKE '%pistón%')
AND activo = 1
```

CATEGORÍAS

1. GET /api/categorias

Propósito: Listar todas las categorías

Response (200):

```
{
  "categorias": [
    {
      "idCategoria": 1,
      "nombre": "Repuestos de Motor",
      "descripcion": "Pistones, bielas, válvulas...",
      "activo": true
    }
  ]
}
```

---

2. POST /api/categorias

Propósito: Crear nueva categoría

Request Body:

```
{
  "nombre": "Accesorios",
  "descripcion": "Espejos, manubrios, etc."
}
```

Validación: Nombre único

---

3. PUT /api/categorias/:id

Propósito: Actualizar categoría

---

4. DELETE /api/categorias/:id

Propósito: Eliminar categoría (soft delete)

Restricción: No se puede eliminar si tiene productos asociados

---

5. GET /api/categorias/:id/productos

Propósito: Listar productos de una categoría

Response (200):

```
{
  "productos": [ ... ]
}
```

🏢 PROVEEDORES

1. GET /api/proveedores

2. POST /api/proveedores

3. PUT /api/proveedores/:id

4. DELETE /api/proveedores/:id

5. GET /api/proveedores/:id/productos

Estructura similar a Categorías

🛒 VENTAS

1. POST /api/ventas (Transacción Compleja)

Propósito: Registrar venta completa con múltiples productos

Request Body:

```
{
  "productos": [
    {
      "idProducto": 1,
      "cantidad": 2,
      "precioUnitario": 85.50
    },
    {
      "idProducto": 5,
      "cantidad": 1,
      "precioUnitario": 150.00
    }
  ],
  "tipoPago": "Efectivo",
  "observaciones": "Cliente frecuente"
}
```

Proceso (Transacción SQL):

```
await pool.query('START TRANSACTION');

try {
  // 1. Crear venta
  const total = productos.reduce((sum, p) => sum + (p.cantidad *
p.precioUnitario), 0);
  const [resultVenta] = await pool.query(
    'INSERT INTO Venta (idUsuario, total, tipoPago, observaciones) VALUES (?, ?,
?, ?)',
    [idUsuario, total, tipoPago, observaciones]
  );

  // 2. Insertar detalles de venta
  for (const producto of productos) {
    await pool.query(
      'INSERT INTO DetalleVenta (idVenta, idProducto, cantidad, precioUnitario,
subtotal) VALUES (?, ?, ?, ?, ?)',
      [resultVenta.insertId, producto.idProducto, producto.cantidad,
producto.precioUnitario, producto.cantidad * producto.precioUnitario]
    );

    // 3. Actualizar stock
    await pool.query(
      'UPDATE Producto SET stock = stock - ? WHERE idProducto = ?',
      [producto.cantidad, producto.idProducto]
    );

    // 4. Registrar movimiento de inventario
    await pool.query(
      'INSERT INTO MovimientoInventario (idProducto, tipoMovimiento, cantidad,
motivo, idUsuario) VALUES (?, ?, ?, ?, ?)',
      [producto.idProducto, 'Salida', producto.cantidad, `Venta
#${resultVenta.insertId}`, idUsuario]
    );
  }

  await pool.query('COMMIT');
  res.status(201).json({ message: 'Venta registrada', idVenta:
resultVenta.insertId });
} catch (error) {
  await pool.query('ROLLBACK');
  res.status(500).json({ message: 'Error al registrar venta' });
}
```

---

2. GET /api/ventas

Propósito: Listar ventas con paginación

Query Params:

- `?page=1&limit=20`
- `?fecha_inicio=2025-01-01&fecha_fin=2025-01-31`

---

3. GET /api/ventas/:id/detalles

Propósito: Ver detalles completos de una venta

Response (200):

```
{
  "venta": {
    "idVenta": 15,
    "fecha": "2025-01-20T15:30:00.000Z",
    "total": 321.00,
    "tipoPago": "Efectivo",
    "nombreVendedor": "Juan Pérez"
  },
  "detalles": [
    {
      "idProducto": 1,
      "nombreProducto": "Pistón 125cc",
      "cantidad": 2,
      "precioUnitario": 85.50,
      "subtotal": 171.00
    },
    {
      "idProducto": 5,
      "nombreProducto": "Filtro de Aceite",
      "cantidad": 1,
      "precioUnitario": 150.00,
      "subtotal": 150.00
    }
  ]
}
```

DASHBOARD

1. GET /api/dashboard/estadisticas

Propósito: Obtener métricas principales

Response (200):

```
{
  "totalProductos": 125,
  "totalCategorias": 15,
```

```
  "totalProveedores": 8,
  "ventasHoy": 15,
  "ventasMes": 320,
  "ingresoHoy": 4850.50,
  "ingresoMes": 125680.00,
  "productosBajoStock": 12,
  "productosAgotados": 3
}
```

---

2. GET /api/dashboard/ventas-recientes

Propósito: Últimas 10 ventas

---

3. GET /api/dashboard/productos-mas-vendidos

Propósito: Top 5 productos

Response (200):

```
{
  "productos": [
    {
      "idProducto": 5,
      "nombre": "Filtro de Aceite",
      "cantidadVendida": 145,
      "totalIngresos": 3625.00
    }
  ]
}
```

---

4. GET /api/dashboard/grafico-ventas-mes

Propósito: Datos para gráfico de líneas (ventas por día del mes)

Response (200):

```
{
  "datos": [
    { "fecha": "2025-01-01", "total": 850.00, "cantidad": 5 },
    { "fecha": "2025-01-02", "total": 1250.00, "cantidad": 8 },
    { "fecha": "2025-01-03", "total": 950.00, "cantidad": 6 }
  ]
}
```

MIDDLEWARE DE AUTENTICACIÓN

```
// index.js líneas 180-195
function verificarToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Token no proporcionado' });
  }

  jwt.verify(token, JWT_SECRET, (err, usuario) => {
    if (err) {
      return res.status(403).json({ message: 'Token inválido o expirado' });
    }
    req.usuario = usuario; // { idUsuario, email, idRol }
    next();
  });
}
```

Uso:

```
app.get('/api/productos', verificarToken, async (req, res) => {
  // req.usuario está disponible aquí
  const idUsuario = req.usuario.idUsuario;
});
```

FORMATO DE RESPUESTAS

Success (200/201):

```
{
  "message": "Operación exitosa",
  "data": { ... }
}
```

Error (400):

```
{
  "message": "Datos inválidos",
  "errors": [
    {
      "field": "email",
      "message": "El email no es válido"
```

```
      }
    ]
}
```

Error (401):

```
{
  "message": "No autorizado - Token requerido"
}
```

Error (403):

```
{
  "message": "Acceso denegado - Permisos insuficientes"
}
```

Error (500):

```
{
  "message": "Error en el servidor",
  "error": "Detalles del error (solo en desarrollo)"
}
```

Siguiente documento: 04-COMPONENTES-FRONTEND.md