Manual Técnico - Sistema de Joyería

Arquitectura del Proyecto

Stack Tecnológico:

```
Frontend: React 18 + TailwindCSS + Recharts
Backend: Node.js + Express + SQLite3
Autenticación: JWT + bcryptjs
Base de Datos: SQLite (desarrollo) / PostgreSQL (producción)
Deploy: Vercel/Railway (gratuito)
```

Estructura del Proyecto:

```
controlsistema/
  client/                  # Frontend React
    src/
      components/     # Componentes reutilizables
      pages/         # Páginas principales
      contexts/      # Estado global (AuthContext)
      utils/         # Utilidades y helpers
      index.css      # Estilos TailwindCSS
    public/           # Assets estáticos
  server/                  # Backend Node.js
    routes/           # Endpoints de la API
    database/         # Configuración y migraciones
    utils/            # Utilidades del servidor
    backups/          # Sistema de backup
  package.json          # Scripts y dependencias
```

Lógica de Negocio

1. Gestión de Inventario

```
// Lógica: Stock Management
class InventoryManager {
  // Al crear producto
  createProduct(product) {
    // 1. Validar datos
    // 2. Generar SKU único
    // 3. Insertar en base de datos
    // 4. Crear movimiento inicial de stock
  }

  // Al registrar venta
  processSale(saleItems) {
```

```
    // 1. Verificar stock disponible
    // 2. Calcular totales
    // 3. Actualizar inventario (transacción)
    // 4. Registrar movimiento de stock
    // 5. Generar factura
  }
}
```

2. Sistema de Ventas

```
// Lógica: Sales Processing
class SalesProcessor {
  processSale(saleData) {
    // 1. Validar productos y cantidades
    // 2. Calcular subtotales y totales
    // 3. Aplicar descuentos (si los hay)
    // 4. Procesar método de pago
    // 5. Actualizar inventario
    // 6. Generar reporte de venta
  }
}
```

3. Reportes y Analytics

```
// Lógica: Reporting Engine
class ReportGenerator {
  generateSalesReport(period) {
    // 1. Consultar ventas del período
    // 2. Agrupar por día/producto/categoría
    // 3. Calcular métricas (total, promedio, crecimiento)
    // 4. Formatear para gráficos
    // 5. Exportar a CSV
  }
}
```

Diseño de Base de Datos

Esquema Principal:

```
-- Tabla de Usuarios
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
```

```
  full_name TEXT NOT NULL,
  role TEXT CHECK(role IN ('administrador', 'vendedor')) NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla de Categorías
CREATE TABLE categories (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT UNIQUE NOT NULL,
  description TEXT,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla de Productos
CREATE TABLE products (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  description TEXT,
  sku TEXT UNIQUE NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  category_id INTEGER,
  stock_quantity INTEGER DEFAULT 0,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (category_id) REFERENCES categories(id)
);

-- Tabla de Ventas
CREATE TABLE sales (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sale_number TEXT UNIQUE NOT NULL,
  customer_name TEXT,
  customer_email TEXT,
  customer_phone TEXT,
  total_amount DECIMAL(10,2) NOT NULL,
  payment_method TEXT CHECK(payment_method IN ('efectivo', 'tarjeta', 'transferencia')),
  user_id INTEGER,
  sale_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Tabla de Items de Venta
CREATE TABLE sale_items (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sale_id INTEGER,
  product_id INTEGER,
```

```
  quantity INTEGER NOT NULL,
  unit_price DECIMAL(10,2) NOT NULL,
  total_price DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (sale_id) REFERENCES sales(id),
  FOREIGN KEY (product_id) REFERENCES products(id)
);

-- Tabla de Movimientos de Stock
CREATE TABLE stock_movements (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  product_id INTEGER,
  movement_type TEXT CHECK(movement_type IN ('entrada', 'salida', 'ajuste')),
  quantity INTEGER NOT NULL,
  previous_stock INTEGER NOT NULL,
  new_stock INTEGER NOT NULL,
  notes TEXT,
  user_id INTEGER,
  movement_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (product_id) REFERENCES products(id),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Índices para Performance:

```
-- Índices para consultas rápidas
CREATE INDEX idx_sales_date ON sales(sale_date);
CREATE INDEX idx_sales_user ON sales(user_id);
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_sku ON products(sku);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_stock_movements_date ON stock_movements(movement_date);
```

Sistema de Autenticación

Flujo de Autenticación:

```
// 1. Login
app.post('/api/auth/login', async (req, res) => {
  const { username, password } = req.body;

  // Buscar usuario
  const user = await findUser(username);
  if (!user) return res.status(401).json({ error: 'Credenciales inválidas' });

  // Verificar contraseña
```

```
  const isValid = await bcrypt.compare(password, user.password_hash);
  if (!isValid) return res.status(401).json({ error: 'Credenciales inválidas' });

  // Generar JWT
  const token = jwt.sign(
    { userId: user.id, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '24h' }
  );

  res.json({ token, user: { id: user.id, username, role: user.role } });
});

// 2. Middleware de Autenticación
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) return res.status(401).json({ error: 'Token requerido' });

  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ error: 'Token inválido' });
    req.user = user;
    next();
  });
};

// 3. Middleware de Autorización
const requireAdmin = (req, res, next) => {
  if (req.user.role !== 'administrador') {
    return res.status(403).json({ error: 'Acceso denegado' });
  }
  next();
};
```

Arquitectura Frontend

Estado Global (Context API):

```
// AuthContext.js
const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));
```

```
  const login = async (credentials) => {
    const response = await axios.post('/api/auth/login', credentials);
    const { token, user } = response.data;

    setToken(token);
    setUser(user);
    localStorage.setItem('token', token);
    axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  };

  const logout = () => {
    setToken(null);
    setUser(null);
    localStorage.removeItem('token');
    delete axios.defaults.headers.common['Authorization'];
  };

  return (
    <AuthContext.Provider value={{ user, token, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Componentes Reutilizables:

```
// Modal Component Pattern
const Modal = ({ isOpen, onClose, title, children }) => {
  if (!isOpen) return null;

  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-
      <div className="bg-white rounded-lg p-6 max-w-md w-full mx-4">
        <div className="flex justify-between items-center mb-4">
          <h2 className="text-xl font-semibold">{title}</h2>
          <button onClick={onClose} className="text-gray-500 hover:text-gray-700">
            <X className="w-6 h-6" />
          </button>
        </div>
        {children}
      </div>
    </div>
  );
};
```

```jsx
// Form Component Pattern
const FormField = ({ label, error, children }) => (
  <div className="mb-4">
    <label className="block text-sm font-medium text-gray-700 mb-2">
      {label}
    </label>
    {children}
    {error && <p className="text-red-600 text-sm mt-1">{error}</p>}
  </div>
);
```

Sistema de Backup

Lógica de Backup Automático:

```jsx
class BackupSystem {
  constructor() {
    this.dbPath = path.join(__dirname, '../database/jewelry_inventory.db');
    this.backupDir = path.join(__dirname, '../backups');
  }

  async createBackup() {
    const timestamp = new Date().toISOString().replace(/[:.]/g, '-');
    const backupPath = path.join(this.backupDir, `backup-${timestamp}.sqlite`);

    // Copiar base de datos
    await fs.copyFile(this.dbPath, backupPath);

    // Comprimir con gzip
    const compressedPath = `${backupPath}.gz`;
    await this.compressFile(backupPath, compressedPath);

    // Limpiar archivo sin comprimir
    await fs.unlink(backupPath);

    return compressedPath;
  }

  async compressFile(inputPath, outputPath) {
    const gzip = util.promisify(zlib.gzip);
    const input = await fs.readFile(inputPath);
    const compressed = await gzip(input);
    await fs.writeFile(outputPath, compressed);
  }
```

```
  async cleanOldBackups(retentionDays = 30) {
    const files = await fs.readdir(this.backupDir);
    const cutoffDate = new Date();
    cutoffDate.setDate(cutoffDate.getDate() - retentionDays);

    for (const file of files) {
      if (file.endsWith('.sqlite.gz')) {
        const filePath = path.join(this.backupDir, file);
        const stats = await fs.stat(filePath);

        if (stats.mtime < cutoffDate) {
          await fs.unlink(filePath);
        }
      }
    }
  }
}
```

Sistema de Reportes

Generación de Reportes:

```
class ReportGenerator {
  async generateSalesReport(period = 'month') {
    const dateFilter = this.getDateFilter(period);

    const query = `
      SELECT
        DATE(s.sale_date) as date,
        COUNT(*) as total_sales,
        SUM(s.total_amount) as total_revenue,
        AVG(s.total_amount) as avg_sale
      FROM sales s
      WHERE s.sale_date >= ?
      GROUP BY DATE(s.sale_date)
      ORDER BY date
    `;

    const sales = await this.db.all(query, [dateFilter]);

    // Formatear para gráficos
    const chartData = sales.map(sale => ({
      date: sale.date,
      sales: sale.total_sales,
      revenue: sale.total_revenue,
```

```
      average: sale.avg_sale
    }));

    return {
      data: chartData,
      summary: {
        totalSales: sales.reduce((sum, s) => sum + s.total_sales, 0),
        totalRevenue: sales.reduce((sum, s) => sum + s.total_revenue, 0),
        averageSale: sales.reduce((sum, s) => sum + s.avg_sale, 0) / sales.length
      }
    };
  }

  async exportToCSV(data, filename) {
    const csv = this.convertToCSV(data);
    const buffer = Buffer.from(csv, 'utf8');

    return {
      buffer,
      filename: `${filename}-${new Date().toISOString().split('T')[0]}.csv`,
      contentType: 'text/csv; charset=utf-8'
    };
  }
}
```

Optimizaciones de Performance

1. Lazy Loading de Componentes:

```
// Cargar componentes solo cuando se necesiten
const Reports = lazy(() => import('./pages/Reports'));
const Backup = lazy(() => import('./pages/Backup'));

// Suspense para mostrar loading
<Suspense fallback={<LoadingSpinner />}>
  <Routes>
    <Route path="/reports" element={<Reports />} />
    <Route path="/backup" element={<Backup />} />
  </Routes>
</Suspense>
```

2. Debouncing en Búsquedas:

```
const useDebounce = (value, delay) => {
  const [debouncedValue, setDebouncedValue] = useState(value);
```

```
  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);

  return debouncedValue;
};

// Uso en búsqueda de productos
const [searchTerm, setSearchTerm] = useState('');
const debouncedSearch = useDebounce(searchTerm, 300);

useEffect(() => {
  if (debouncedSearch) {
    searchProducts(debouncedSearch);
  }
}, [debouncedSearch]);
```

3. Memoización de Componentes:

```
const ProductCard = memo(({ product, onEdit, onDelete }) => {
  return (
    <div className="card">
      <h3>{product.name}</h3>
      <p>${product.price}</p>
      <p>Stock: {product.stock_quantity}</p>
      <button onClick={() => onEdit(product)}>Editar</button>
      <button onClick={() => onDelete(product.id)}>Eliminar</button>
    </div>
  );
});
```

Configuración de Seguridad

1. Rate Limiting:

```
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
```

```
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100, // máximo 100 requests por ventana
  message: 'Demasiadas requests desde esta IP',
  standardHeaders: true,
  legacyHeaders: false,
});

app.use('/api/', limiter);
```

2. Helmet Security Headers:

```
const helmet = require('helmet');

app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      scriptSrc: ["'self'"],
      imgSrc: ["'self'", "data:", "https:"],
    },
  },
}));
```

3. CORS Configuration:

```
const cors = require('cors');

app.use(cors({
  origin: process.env.CORS_ORIGIN || 'http://localhost:3000',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization'],
}));
```

Responsive Design

Breakpoints TailwindCSS:

```
/* Mobile First Approach */
.sidebar {
  @apply fixed inset-y-0 left-0 z-50 w-64 bg-white shadow-large transform transition-transfo
}

/* Desktop: siempre visible */
```

```css
@media (min-width: 1024px) {
  .sidebar {
    @apply translate-x-0;
  }

  .main-content {
    @apply ml-64;
  }
}

/* Mobile: oculto por defecto */
@media (max-width: 1023px) {
  .sidebar {
    @apply -translate-x-full;
  }

  .sidebar-open {
    @apply translate-x-0;
  }
}
```

Componentes Responsivos:

```jsx
const ResponsiveTable = ({ data, columns }) => {
  const [isMobile, setIsMobile] = useState(false);

  useEffect(() => {
    const checkMobile = () => {
      setIsMobile(window.innerWidth < 768);
    };

    checkMobile();
    window.addEventListener('resize', checkMobile);
    return () => window.removeEventListener('resize', checkMobile);
  }, []);

  if (isMobile) {
    return (
      <div className="space-y-4">
        {data.map((item, index) => (
          <div key={index} className="card p-4">
            {columns.map(column => (
              <div key={column.key} className="flex justify-between mb-2">
                <span className="font-medium">{column.label}:</span>
                <span>{item[column.key]}</span>
              </div>
```

```
            ))}
          </div>
        ))}
      </div>
    );
  }

  return (
    <table className="w-full">
      {/* Tabla normal para desktop */}
    </table>
  );
};
```

Testing Strategy

1. Unit Tests (Jest):

```
// tests/inventory.test.js
describe('InventoryManager', () => {
  test('should create product with valid data', async () => {
    const product = {
      name: 'Anillo de Diamante',
      price: 1500,
      stock: 5
    };

    const result = await inventoryManager.createProduct(product);

    expect(result).toHaveProperty('id');
    expect(result.sku).toMatch(/^SKU-\d{6}$/);
    expect(result.stock_quantity).toBe(5);
  });

  test('should prevent sale with insufficient stock', async () => {
    const saleData = {
      items: [{ productId: 1, quantity: 10 }]
    };

    await expect(salesProcessor.processSale(saleData))
      .rejects.toThrow('Stock insuficiente');
  });
});
```

2. Integration Tests:

```
// tests/api.test.js
describe('Sales API', () => {
  test('POST /api/sales should create sale', async () => {
    const saleData = {
      items: [{ productId: 1, quantity: 1 }],
      customerName: 'Juan Pérez',
      paymentMethod: 'efectivo'
    };

    const response = await request(app)
      .post('/api/sales')
      .set('Authorization', `Bearer ${token}`)
      .send(saleData);

    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('sale_number');
  });
});
```

Deployment Strategy

1. Environment Configuration:

```
// config/environment.js
const config = {
  development: {
    database: './database/jewelry_inventory.db',
    port: 5001,
    corsOrigin: 'http://localhost:3000',
    jwtSecret: 'dev-secret-key'
  },
  production: {
    database: process.env.DATABASE_URL,
    port: process.env.PORT || 5001,
    corsOrigin: process.env.CORS_ORIGIN,
    jwtSecret: process.env.JWT_SECRET
  }
};

module.exports = config[process.env.NODE_ENV || 'development'];
```

2. Build Process:

```
{
  "scripts": {
    "build": "cd client && npm run build",
    "start": "cd server && npm start",
    "dev": "concurrently \"npm run server\" \"npm run client\"",
    "test": "jest",
    "lint": "eslint .",
    "deploy": "npm run build && npm run test && npm run start"
  }
}
```

Métricas y Monitoreo

1. Performance Metrics:

```
// middleware/performance.js
const performanceMiddleware = (req, res, next) => {
  const start = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - start;
    console.log(`${req.method} ${req.path} - ${duration}ms - ${res.statusCode}`);

    // Enviar métricas a servicio de monitoreo
    if (duration > 1000) {
      console.warn(`Slow request: ${req.path} took ${duration}ms`);
    }
  });

  next();
};
```

2. Error Tracking:

```
// middleware/errorHandler.js
const errorHandler = (err, req, res, next) => {
  console.error('Error:', {
    message: err.message,
    stack: err.stack,
    url: req.url,
    method: req.method,
    user: req.user?.id
  });
```

```
  res.status(err.status || 500).json({
    error: process.env.NODE_ENV === 'production'
      ? 'Error interno del servidor'
      : err.message
  });
};
```

Lecciones Aprendidas

1. **Decisiones Técnicas Correctas:**
   - **SQLite para desarrollo:** Fácil setup y portabilidad
   - **JWT para autenticación:** Stateless y escalable
   - **Context API para estado:** Simple y efectivo para este tamaño
   - **TailwindCSS:** Desarrollo rápido y consistente

2. **Optimizaciones Futuras:**
   - **Implementar Redis** para caché de reportes
   - **Migrar a PostgreSQL** para producción
   - **Agregar WebSockets** para actualizaciones en tiempo real
   - **Implementar PWA** para funcionalidad offline

3. **Escalabilidad:**
   - **Microservicios** cuando crezca el negocio
   - **Base de datos distribuida** para múltiples ubicaciones
   - **API Gateway** para manejar múltiples clientes
   - **CDN** para assets estáticos

Conclusión

Este proyecto demuestra una arquitectura sólida para un sistema de inventario pequeño-mediano:

Fortalezas:

- **Arquitectura modular** y mantenible
- **Seguridad implementada** desde el inicio
- **Responsive design** para todos los dispositivos
- **Sistema de backup** automático
- **Reportes completos** con exportación

Tecnologías Utilizadas:

- **Frontend:** React 18, TailwindCSS, Recharts, Axios
- **Backend:** Node.js, Express, SQLite3, JWT
- **Herramientas:** Git, npm, nodemon, concurrently
- **Deploy:** Vercel/Railway (gratuito)

Próximos Pasos:

1. **Implementar tests** automatizados
2. **Agregar CI/CD** pipeline
3. **Optimizar performance** con lazy loading
4. **Implementar PWA** features
5. **Agregar analytics** y métricas

¡El sistema está listo para crecer con tu negocio!