Sistema de Gestión de Precios con IA

Nacimiento del problema y Resumen Ejecutivo:

Un conocido abre una pequeña tienda de alimentación y se encuentra con una gran dificultad para mantener un registro del precio de todos sus productos, así como el desconocimiento de como poner precio a alguno de estos

De ahí nace PriceStockChecker. Una aplicación web full-stack diseñada para optimizar la estrategia de precios en comercios minoristas mediante inteligencia artificial. Combina tecnologías modernas de desarrollo web con modelos de Inteligencia Artificial para proporcionar recomendaciones de precios contextualizadas y específicas por categoría de producto.

Arquitectura del Sistema

Patrones Arquitectónicos Implementados

1. **MVC (Modelo-Vista-Controlador):**
   Se adoptó el patrón MVC para estructurar la aplicación de forma clara y modular. El **modelo** se basa en una base de datos **SQLite** con un esquema relacional definido. La **vista** se implementó mediante **plantillas EJS** con **CSS personalizado** para una presentación visual simple. El **controlador** se organizó usando **rutas Express.js**, asegurando una lógica de flujo clara y mantenible.

2. **Middleware Pattern:**
   Se incorporaron middleware para manejar funcionalidades transversales esenciales. Esto incluye **autenticación y autorización**, **validación de datos de entrada** y un sistema de **manejo de errores centralizado**. Esta estructura mejora la seguridad, la calidad de los datos y la estabilidad del sistema ante posibles fallos.

3. **Repository Pattern:**
   Se implementó este patrón para **abstraer la capa de acceso a datos**, promoviendo un código más limpio y desacoplado. Esta decisión facilita la **migración entre distintos motores de base de datos** en el futuro y mejora la escalabilidad y el mantenimiento de la aplicación.

Tecnologías Implementadas

Backend Stack

Tecnología	Versión	Justificación de Elección
Node.js	18.x+	Runtime asíncrono ideal para I/O operations
Express.js	4.x	Minimalista, flexible, ecosistema robusto
SQLite3	5.x	Base de datos embebida, cero configuración
Multer	1.x	Manejo de uploads de archivos eficiente
Axios	1.x	Cliente HTTP para APIs externas

Frontend Stack

Tecnología	Versión	Ventajas
EJS	3.x	Templating server-side, fácil aprendizaje
CSS3	-	Flexbox, Grid, animaciones nativas
JavaScript ES6+	-	Async/await, fetch API, modern syntax

IA Stack

Tecnología	Uso	Razón de Elección
OpenRouter API	Gateway unificado	Acceso a múltiples modelos LLM
DeepSeek R1	Modelo principal	Balance costo-rendimiento

Base de Datos

Esquema Relacional Optimizado

```
CREATE TABLE productos (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nombre TEXT NOT NULL CHECK(LENGTH(nombre) >= 2),
  marca TEXT NOT NULL,
  precio_compra REAL NOT NULL CHECK(precio_compra > 0),
  precio_actual REAL CHECK(precio_actual >= 0),
  calidad TEXT CHECK(calidad IN ('alta', 'media', 'baja')) NOT NULL,
  urgencia TEXT CHECK(urgencia IN ('muy alta', 'media', 'baja')) NOT NULL,
  categoria TEXT CHECK(categoria IN ('Fruta/Verdura', 'Bebida', 'Alimento',
'Otros')),
  foto TEXT,
  precio_sugerido REAL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE historico_precios (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  producto_id INTEGER REFERENCES productos(id) ON DELETE CASCADE,
  fecha DATETIME DEFAULT CURRENT_TIMESTAMP,
  precio_actual REAL,
  precio_sugerido REAL,
  diferencia REAL GENERATED ALWAYS AS (precio_sugerido - precio_actual)
VIRTUAL
```

Índices para Optimización

```
CREATE INDEX idx_productos_nombre ON productos(nombre COLLATE NOCASE);
CREATE INDEX idx_productos_categoria ON productos(categoria);
CREATE INDEX idx_productos_precio ON productos(precio_actual);
CREATE INDEX idx_historico_producto ON historico_precios(producto_id);
CREATE INDEX idx_historico_fecha ON historico_precios(fecha DESC);
```

Flujo de Procesamiento IA

1. Preprocesamiento: Normalización y validación de datos
2. Selección de Template: Según categoría del producto
3. Enriquecimiento Contextual: Mercado local + temporada
4. Postprocesamiento: Validación numérica y formatos
5. Auditoría: Registro en histórico para mejora continua

Características Principales

1. Sistema de Gestión de Productos Completo

CRUD Avanzado:

- Creación con validación de datos en tiempo real
- Edición con histórico de cambios
- Eliminación con transacciones ACID
- Upload de imágenes con optimización

Validaciones Implementadas:

```javascript
const validaciones = {
  precio: value => value > 0 && value < 10000,
  nombre: value => value.length >= 2 && value.length <= 100,
  calidad: value => ['alta', 'media', 'baja'].includes(value),
  urgencia: value => ['muy alta', 'media', 'baja'].includes(value)
};
```

2. Motor de Búsqueda Inteligente

Algoritmo de Búsqueda:

- Búsqueda por aproximación fonética
- Normalización Unicode (remove diacríticos)
- Indexación en memoria para resultados rápidos
- Paginación server-side preparada

```javascript
function normalizarBusqueda(texto) {
  return texto
    .normalize('NFD')
    .replace(/[\u0300-\u036f]/g, '')
    .toLowerCase()
    .trim();
}
```

3. Sistema de Categorización Inteligente

Taxonomía de Productos:

- 4 categorías principales expandibles
- Sistema de etiquetas preparado
- Metadata extensible para futuras necesidades

Análisis de Seguridad

Capas de Protección Implementadas

1. Validación de Entrada:
    - Sanitización XSS
    - Validación de tipos de datos
    - Límites numéricos razonables
2. Protección de Archivos:
    - Whitelist de extensiones
    - Scan de malware (preparado)
    - Límite de tamaño de upload
3. Seguridad de Base de Datos:
    - Prepared statements
    - Escapado de queries
    - Validación de esquema

Hardening de Express.js

```
javascript
// Configuración de seguridad
app.disable('x-powered-by');
app.use(helmet());
app.use(cors({
  origin: process.env.NODE_ENV === 'production' ? 'https://midominio.com' :
'*'

}));
```

Optimización y Rendimiento

Estrategias de Caching

```javascript
const cache = new NodeCache({
  stdTTL: 300, // 5 minutos
  checkperiod: 60
});

// Cache de productos frecuentes
app.get('/producto/:id', (req, res) => {
  const cacheKey = `producto_${req.params.id}`;
  const cached = cache.get(cacheKey);

  if (cached) return res.json(cached);

  // ... lógica normal y luego cache.set(cacheKey, resultado)

});
```

Optimización de Base de Datos

1. Connection Pooling: Reutilización de conexiones
2. Query Optimization: Índices compuestos
3. Lazy Loading: Carga bajo demanda de imágenes
4. Paginación: LIMIT/OFFSET para grandes datasets

Metricas de Rendimiento

Operación	Tiempo Esperado	Optimizaciones Aplicadas
Búsqueda	< 100ms	Índices + cache en memoria
Cálculo IA	2-5s	Prompt optimizado + modelo eficiente
Carga página	< 1s	Assets optimizados + CDN ready

Roadmap de Expansión

Fase 1: Mejoras Inmediatas (1-3 meses)

Autenticación y Autorización:

- Sistema de roles (admin, usuario, invitado)
- OAuth2 con Google/GitHub
- JWT con refresh tokens

Panel Administrativo:

- Dashboard con métricas de negocio
- Reportes de rentabilidad
- Análisis de tendencias de precios

Fase 2: Expansión Funcional (3-6 meses)

Multi-Tienda:

- Soporte para múltiples locales
- Sincronización en tiempo real
- Gestión centralizada

Integración APIs Externas:

- Precios de competencia via scraping
- Datos de mercado en tiempo real
- Integración con sistemas ERP

Mobile App:

- React Native para iOS/Android
- Funcionalidad offline
- Sincronización background

Fase 3: Inteligencia Avanzada (6-12 meses)

Machine Learning Propio:

- Modelos personalizados por tipo de producto

- Predictive analytics para demanda
- Sistema de recomendaciones automático

Blockchain para Auditoría:

- Histórico de precios inmutable
- Transparencia para regulaciones
- Smart contracts para acuerdos

IoT Integration:

- Sensores de inventario automático
- Precios dinámicos según stock
- Integración con balanzas inteligentes

Conclusiones Técnicas

Fortalezas del Sistema Actual

1. Arquitectura Sólida: Base escalable y mantenible
2. Stack Moderno: Tecnologías actualizadas y con buen soporte
3. IA Integrada: Diferenciador competitivo inmediato
4. Performance: Optimizado para el caso de uso específico
5. Extensibilidad: Preparado para crecimiento futuro

Decisiones Técnicas Clave

1. SQLite sobre PostgreSQL: Elección consciente para MVP, con migración preparada
2. EJS sobre React: Decision basada en simplicidad y time-to-market
3. OpenRouter sobre API directa: Abstracción que permite cambiar modelos fácilmente

Recomendaciones de Evolución

1. Microservicios: Separar módulos cuando crezca la complejidad
2. TypeScript: Adoptar para mejor mantenibilidad a largo plazo
3. Testing Suite: Implementar Jest + Cypress para cobertura completa
4. CI/CD Pipeline: Automatizar despliegues y testing