Pontificia Universidad Javeriana
Departamento de Ingeniería De Sistemas

Laboratorio diseño
laboratorio

Fundamentos de software
Profesor Luis Moreno Sandoval

Desarrollado por
Juan Felipe Romero Ortiz

---

1. Diagrama ECB

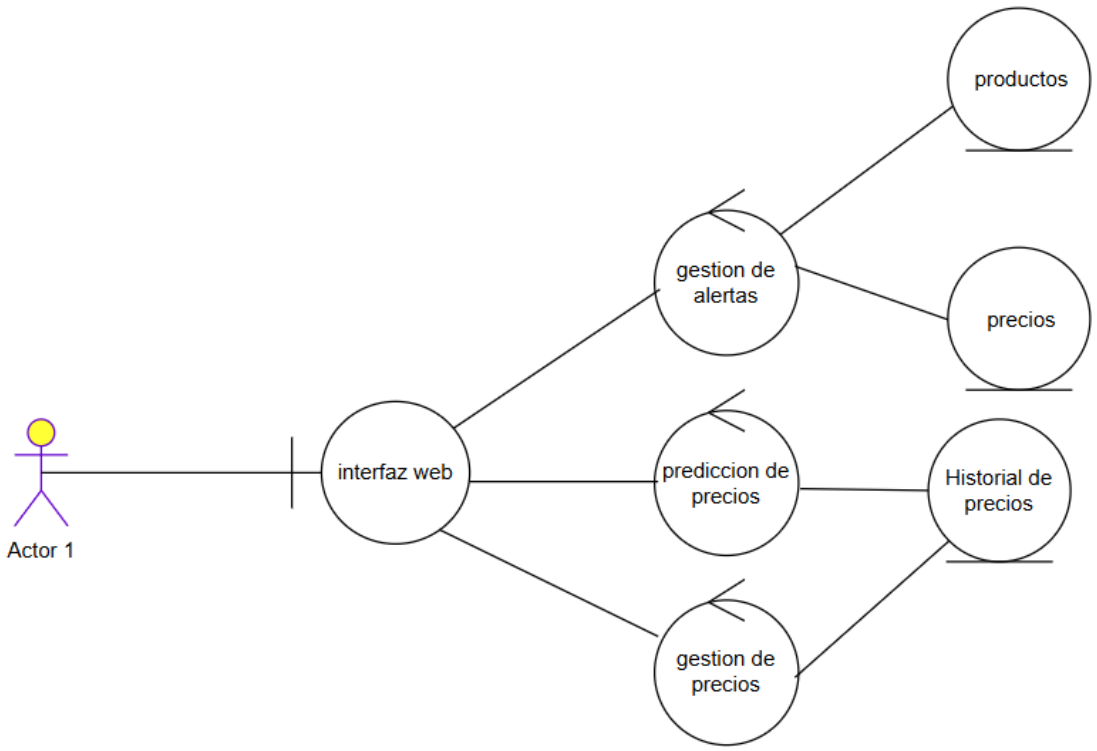

Figura 1: Diagrama ECB

2. Descripción del Diagrama ECB

El diagrama ECB (Event-Control-Boundary) presentado muestra la arquitectura fundamental del sistema FEED-Back, diseñado para el monitoreo y comparación de precios de alimentos. A continuación se detalla cada componente y su interrelación:

2.1. Elementos del Diagrama

2.1.1. Entidades (Modelo de Datos)

- **Productos**: Representa la base de datos central de todos los artículos alimenticios monitoreados por el sistema. Contiene atributos como:
  - Identificador único
  - Categoría (perecederos, granos, lácteos, etc.) de sostenibilidad (huella ecológica, origen local)
- **Precios**: Almacena la información dinámica de costos asociada a cada producto, incluyendo:
  - Valor actual
  - Histórico de fluctuaciones
  - Establecimiento asociado (supermercado o mercado local)

2.1.2. Componentes de Control (Lógica de Negocio)

- **Gestión de Precios**: Módulo central que:
  - Actualiza los precios en tiempo real mediante APIs
  - Normaliza datos de diferentes fuentes (supermercados, mercados locales)
  - Implementa la lógica de comparación entre establecimientos
- **Predicción de Precios**: Componente inteligente que:
  - Utiliza modelos estadísticos y de IA (como ARIMA o redes neuronales)
  - Genera proyecciones a corto plazo basadas en tendencias históricas
  - Considera variables externas (estacionalidad, eventos climáticos)
- **Gestión de Alertas**: Subsistema responsable de:
  - Monitorear umbrales de cambio de precios configurados por usuarios
  - Disparar notificaciones push/email cuando se detectan variaciones significativas
  - Priorizar alertas según preferencias del usuario (productos favoritos)

2.1.3. Interfaces (Boundary)

- **Interfaz Web**: Punto de contacto principal con los usuarios finales que:
  - Presenta comparativas visuales de precios
  - Permite configurar alertas personalizadas
  - Muestra recomendaciones de productos sostenibles
- **Historial de Precios**: Módulo de visualización que:
  - Genera gráficos temporales (semanales, mensuales, anuales)
  - Permite exportar datos para análisis externo
  - Incluye herramientas de filtrado por categoría/productor

2.2. Flujos y Relaciones Clave

- La **Interfaz Web** consume servicios tanto de **Gestión de Precios** (para datos actuales) como de **Predicción de Precios** (para tendencias futuras).
- El módulo de **Gestión de Alertas** depende directamente de las actualizaciones en la entidad **Precios** y se integra con la **Interfaz Web** para notificaciones en tiempo real.
- El **Historial de Precios** se alimenta tanto de datos históricos almacenados como de proyecciones generadas por el componente de predicción.
- La entidad **Productos** sirve como referencia maestra para todos los demás componentes, asegurando consistencia en la información básica de los artículos.

2.3. Arquitectura y Diseño

La estructura sigue un patrón MVC (Modelo-Vista-Controlador) adaptado:

- **Modelo**: Entidades (Productos, Precios)
- **Controlador**: Componentes de gestión y predicción
- **Vista**: Interfaces web e históricos

Se observa una clara separación de responsabilidades que permite:

- Escalabilidad horizontal (añadir nuevos proveedores de datos sin modificar el core)
- Mantenibilidad (actualizar componentes de IA independientemente de la interfaz)
- Flexibilidad para incorporar nuevos tipos de análisis (ej: huella de carbono por producto)

3. diagrama de clases

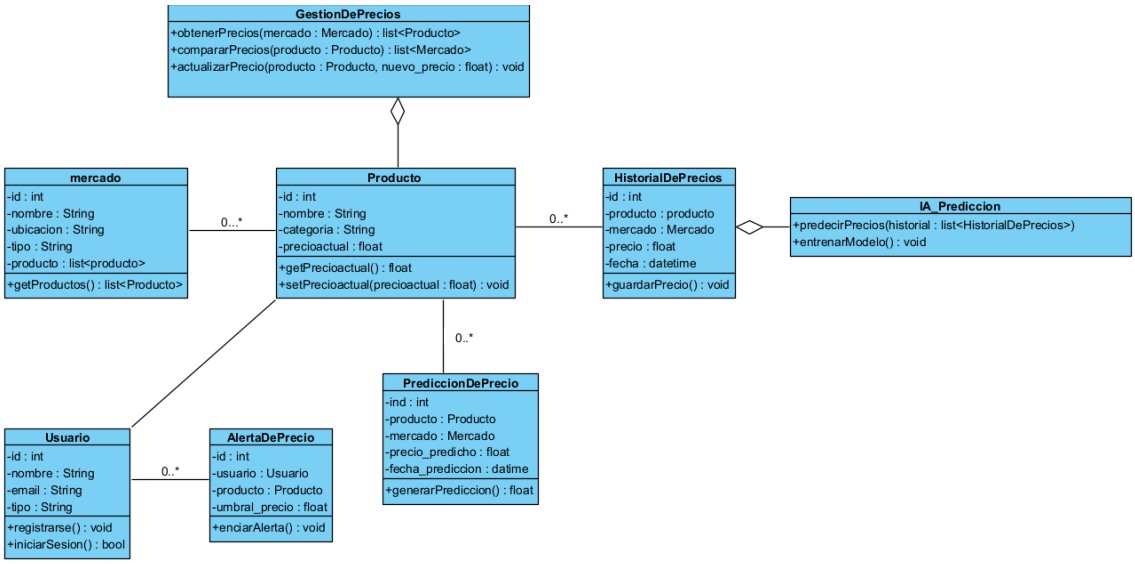

Figura 2: Diagrama de clases

4. Descripción del Diagrama de Clases

El diagrama de clases presentado modela el sistema FEED-Back para la gestión y predicción de precios de alimentos, mostrando una estructura robusta con relaciones bien definidas. A continuación se analiza cada componente:

4.1. Clases Principales

4.1.1. GestiónDePrecios (Clase Controladora)

- **Responsabilidad**: Coordina las operaciones centrales del sistema
- **Atributos**: No muestra atributos propios (posiblemente singleton)
- **Métodos**:
  - `obtenerPrecios(mercado:Mercado)`: Devuelve lista de productos por establecimiento
  - `compararPrecios(producto:Producto)`: Compara precios entre mercados
  - `actualizarPrecio(...)`: Actualiza valores en el sistema
- **Relaciones**: Interactúa con `Mercado`, `Producto` e indirectamente con `HistorialDePrecios`

4.1.2. Mercado (Entidad)

- **Atributos**:
  - `id`, `nombre`, `ubicación`, `tipo` (supermercado/mercado local)
  - `productos:List<Producto>` (relación 1-N)
- **Métodos**:
  - `getProductos()`: Obtiene productos disponibles
- **Relaciones**: Contiene múltiples `Producto`, asociado a `HistorialDePrecios`

4.1.3. Producto (Entidad)

- **Atributos**:
  - `id`, `nombre`, `categoría`
  - `precioActual:float` (valor dinámico)
- **Métodos**:
  - `getPrecioActual()`, `setPrecioActual()`: Getters/setters básicos
- **Relaciones**: Relacionado con `Mercado`, `HistorialDePrecios`, `AlertaDePrecio`

4.1.4. HistorialDePrecios (Entidad Histórica)

- **Atributos**:
  - `id`, `precio:float`, `fecha:DateTime`
  - Referencias a `Producto` y `Mercado`
- **Métodos**:
  - `guardarPrecio()`: Persiste registros históricos

4.2. Clases de Inteligencia Artificial

4.2.1. IA_Prediccion (Servicio)

- **Responsabilidad**: Generar proyecciones de precios
- **Métodos**:
  - `predecirPrecios(historial:List<HistorialDePrecios>)`: Ejecuta modelos predictivos
  - `entrenarModelo()`: Actualiza algoritmos de IA
- **Relaciones**: Consume `HistorialDePrecios`, genera `PrediccionDePrecio`

4.2.2. PrediccionDePrecio (Entidad Resultado)

- **Atributos**:
  - `precioPredicho:float`, `fechaPrediccion:DateTime`
  - Relaciones con `Producto` y `Mercado`

4.3. Clases de Usuario

4.3.1. Usuario (Entidad)

- **Atributos**:
  - `id`, `nombre`, `email`, `tipo` (posibles roles)
- **Métodos**:
  - `registrarse()`, `iniciarSesion()`: Gestión de autenticación

4.3.2. AlertaDePrecio (Entidad de Configuración)

- **Atributos**:
  - `umbralPrecio:float` (valor para disparar notificaciones)
  - Relaciones con `Usuario` y `Producto`
- **Métodos**:
  - `enviarAlerta()`: Dispara mecanismos de notificación

5. Análisis de Relaciones

- **Agregaciones**:
  - `Mercado` contiene múltiples `Producto`
  - `Usuario` tiene asociadas varias `AlertaDePrecio`
- **Dependencias**:
  - `IA_Prediccion` depende de `HistorialDePrecios`
  - `GestiónDePrecios` depende de `Mercado` y `Producto`
- **Patrones Detectados**:
  - Estrategia (para algoritmos de predicción intercambiables)
  - Observer (en el sistema de alertas)
  - Factory (posible creación de diferentes tipos de mercados)

6. Conclusiones de Diseño

- **Cobertura de Requisitos**:
  - Implementa completamente las funcionalidades descritas en el canvas (comparación, alertas, predicción)
  - Faltaría integrar clases para métricas de sostenibilidad (huella ecológica)
- **Mejoras Propuestas**:
  - Añadir clase `Sostenibilidad` asociada a `Producto`

- Incluir `ProveedorLocal` como especialización de `Mercado`
- Implementar interfaz `Notificador` para múltiples canales de alerta

- **Escalabilidad**:
  - Diseño modular permite añadir nuevos tipos de análisis
  - Relaciones claras facilitan extensión a nuevos mercados o productos

7. Diagrama De Componentes

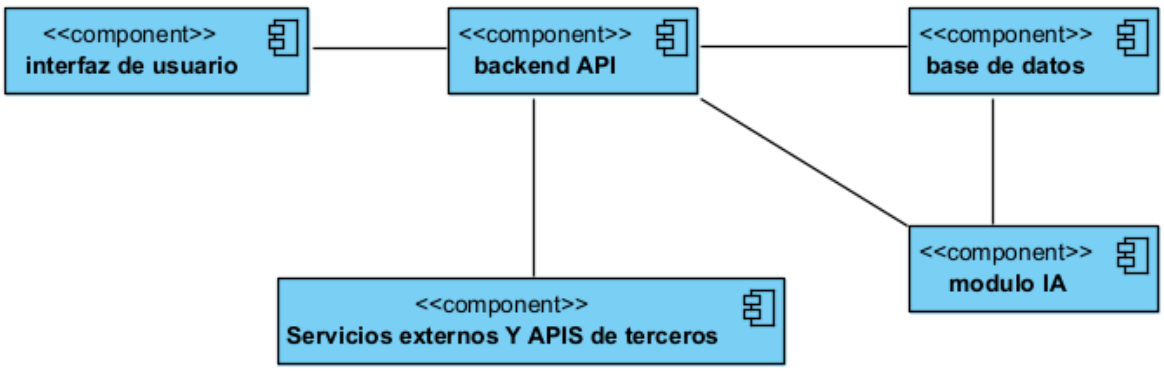

Figura 3: Diagrama De Componentes

8. Descripción del Diagrama de Componentes

El diagrama de componentes presenta la arquitectura modular del sistema FEED-Back, mostrando los bloques fundamentales y sus interacciones. Este análisis se enfoca en cada componente y su rol dentro del ecosistema del sistema.

8.1. Componentes Principales

8.1.1. Interfaz de Usuario

- **Función**: Capa de presentación y punto de interacción con los usuarios finales
- **Subcomponentes implícitos**:
  - Módulo web (posiblemente React/Angular)
  - Aplicación móvil (Flutter/React Native)
  - Panel administrativo
- **Responsabilidades**:
  - Mostrar comparativas de precios
  - Capturar preferencias de alertas
  - Visualizar predicciones y recomendaciones
- **Interfaces**:
  - Consume `Backend API` mediante REST/GraphQL
  - Soporta múltiples formatos de visualización (gráficos, tablas, mapas)

8.1.2. Backend API

- **Función**: Cerebro del sistema que orquesta todas las operaciones
- **Subcomponentes lógicos**:
  - API Gateway (manejo de rutas)
  - Microservicio de gestión de precios
  - Microservicio de alertas
  - Servicio de autenticación
- **Responsabilidades**:
  - Procesar solicitudes de la interfaz
  - Coordinar operaciones entre componentes
  - Gestionar seguridad y acceso
- **Interfaces**:
  - Expone endpoints RESTful
  - Se comunica con todos los demás componentes

8.1.3. Base de Datos

- **Función**: Almacenamiento persistente de toda la información
- **Posible implementación**:
  - PostgreSQL (para datos transaccionales)
  - MongoDB (para datos no estructurados)
  - Redis (caché)
- **Estructuras clave**:
  - Tablas de productos y precios
  - Históricos temporales
  - Perfiles de usuario
- **Patrones de acceso**:
  - Acceso exclusivo mediante `Backend API`
  - Replicación para alta disponibilidad

8.1.4. Módulo IA

- **Función**: Procesamiento inteligente y predictivo
- **Subcomponentes**:
  - Modelo de predicción de precios
  - Motor de recomendaciones
  - Sistema de detección de anomalías
- **Tecnologías probables**:
  - Python con TensorFlow/PyTorch
  - Servicios de AWS SageMaker o equivalentes

- **Flujos principales**:
  - Entrenamiento con datos históricos
  - Generación de predicciones bajo demanda
  - Análisis de patrones de consumo

8.1.5. Servicios Externos y APIs de Terceros

- **Función**: Integración con ecosistemas externos
- **Ejemplos concretos**:
  - APIs de supermercados (precios actuales)
  - Servicios de geolocalización
  - Datos climáticos (para modelos predictivos)
  - APIs de huella de carbono
- **Patrones de integración**:
  - Conexiones sincrónicas (REST)
  - Consumo asíncrono (colas de mensajes)
  - Adaptadores para formatos diversos

9. Interacciones entre Componentes

- **Flujo típico de consulta**:
  1. `Interfaz de Usuario` → `Backend API`: Solicitud de precios
  2. `Backend API` → `Base de Datos`: Consulta información básica
  3. `Backend API` → `Módulo IA`: Solicita predicciones
  4. `Backend API` → `Servicios Externos`: Actualiza datos en tiempo real
  5. `Backend API` → `Interfaz de Usuario`: Devuelve respuesta consolidada
- **Flujo de actualización**:
  1. `Servicios Externos` → `Backend API`: Notifica cambios
  2. `Backend API` → `Base de Datos`: Actualiza registros
  3. `Backend API` → `Módulo IA`: Reentrena modelos si es necesario
  4. `Backend API` → `Interfaz de Usuario`: Push notifications

10. Evaluación Arquitectónica

10.1. Fortalezas del Diseño

- **Modularidad**: Componentes desacoplados permiten actualizaciones independientes
- **Escalabilidad**: Cada componente puede escalar horizontalmente según demanda
- **Resistencia**: Fallos en un módulo no colapsan todo el sistema

10.2. Recomendaciones de Mejora

- Añadir componente de **Analítica** separado para procesamiento de métricas
- Incluir **API Management** para mejor gobierno de interfaces
- Considerar **Bus de Eventos** para comunicaciones asíncronas

10.3. Consistencia con el Canvas

- El `Módulo IA` soporta la "Ventaja Injusta"de predicciones
- `Servicios Externos` permiten el Çomparativo de precios"
- La arquitectura facilita los Çanales"multi-plataforma

11. Diagrama de despliegue

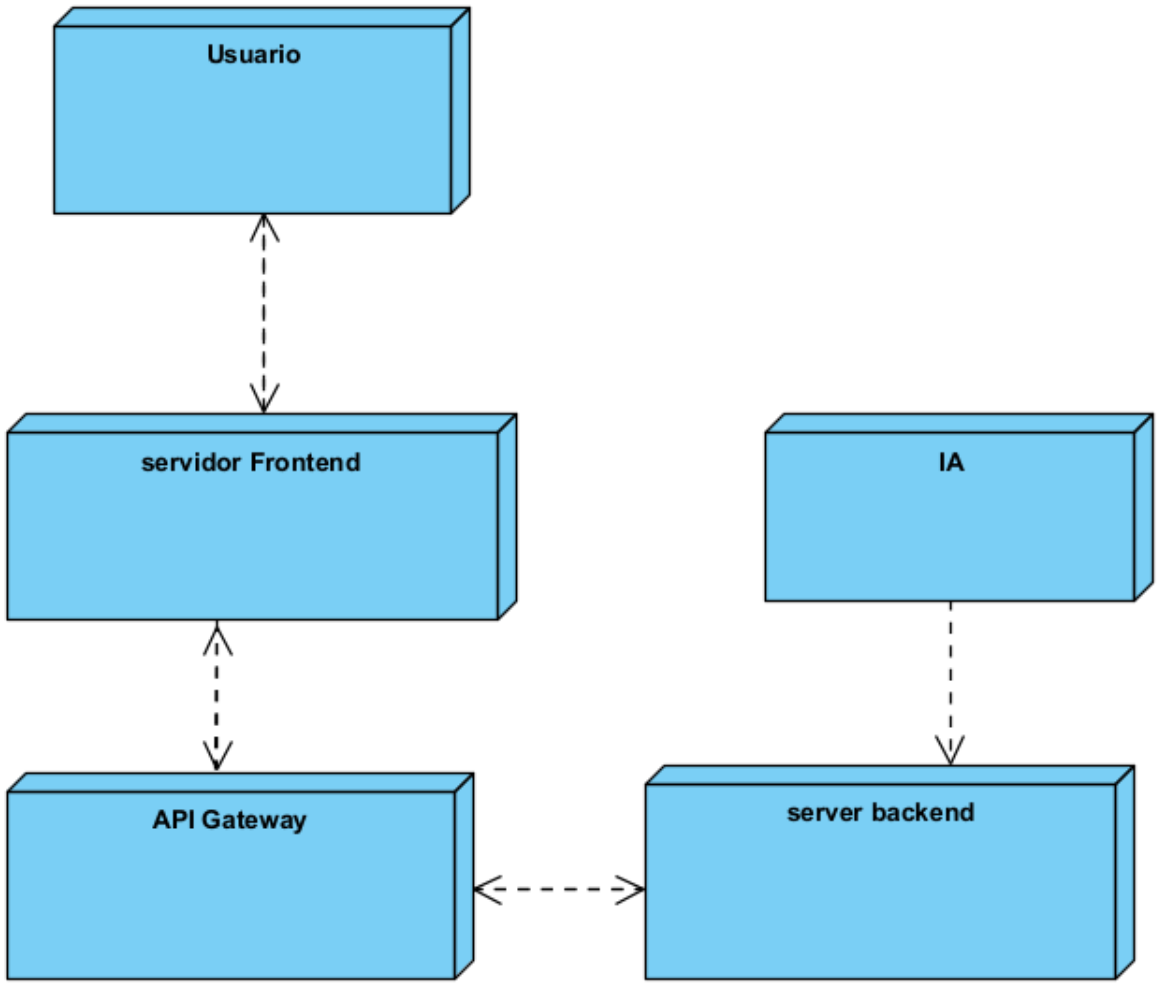

Figura 4: Diagrama de Despliegue

12. Descripción General

El diagrama muestra una arquitectura básica en tres capas:

- **Capa de Presentación**: Interfaz para usuarios finales
- **Capa de Lógica**: Procesamiento central y gestión de APIs
- **Capa de Inteligencia**: Módulos especializados en IA

13. Componentes Principales

13.1. Nodos Identificados

- **Frontend**: Servidor web/móvil que entrega la interfaz
- **Backend**: Servidor principal con la lógica de negocio
- **IA**: Servidor para modelos predictivos (opcional en fase inicial)
- **API Gateway**: Punto único de entrada para las APIs

14. Flujo Básico

1. Usuarios interactúan con el Frontend
2. Frontend consulta al Backend via API Gateway
3. Backend decide si requiere procesamiento de IA
4. Respuestas regresan al usuario por el mismo camino

15. Consideraciones Clave

- **Escalabilidad**: Diseño modular para crecer según demanda
- **Tecnologías**: Por definir (se evaluarán opciones simples iniciales)
- **Seguridad**: Protección básica HTTPS y autenticación

16. Próximos Pasos

- Definir stack tecnológico mínimo viable
- Estimar costos de infraestructura inicial
- Priorizar componentes para desarrollo

17. Conclusiones Generales

- **Arquitectura viable**: Los diagramas presentan una base sólida para desarrollar el sistema FEED-Back, cubriendo los requisitos principales del canvas.
- **Modularidad**: La separación en componentes claros (frontend, backend, IA) permitirá desarrollo paralelo y escalabilidad futura.
- **Flexibilidad**: El diseño permite adaptarse a diferentes tecnologías según disponibilidad y necesidades del equipo.

18. Puntos Fuertes

- **Enfoque en IA**: La integración de predicciones de precios como componente independiente ofrece una ventaja competitiva clara.
- **Comunicación estandarizada**: El uso de API Gateway simplificará la integración entre módulos y con servicios externos.
- **User-centric**: La separación clara de la interfaz garantiza experiencia de usuario consistente.

19. Retos Identificados

- **Complejidad de integración**: La conexión con múltiples APIs de terceros (supermercados, mercados) requerirá gestión cuidadosa.
- **Curva de aprendizaje**: El componente de IA necesitará especialización o colaboración con expertos.
- **Balance costo-rendimiento**: La infraestructura necesaria para tiempo real deberá optimizarse.

20. Recomendaciones

- **Enfoque incremental**: Implementar primero un MVP sin IA para validar el modelo.
- **Pruebas tempranas**: Realizar prototipos de integración con al menos 2 APIs de supermercados.
- **Monitoreo continuo**: Establecer métricas desde el inicio para evaluar rendimiento real.