Taller Electiva Profesional - *Juanesteban Ruiz Carrillo* 03/10/2025

Plataforma Agrícola (Red digital para productores agrícolas)

I. Contexto

El Ministerio de Agricultura busca conectar a pequeños y medianos productores agrícolas con compradores y distribuidores a través de una plataforma digital nacional. Actualmente, los procesos de comercialización son manuales y poco trazables. Se propone el desarrollo de una plataforma distribuida basada en microservicios que permita gestionar inventarios, contratos de compra, transporte y certificaciones.

Objetivo del proyecto

Construir una arquitectura basada en microservicios que soporte el crecimiento de usuarios, la integración con entidades externas y la seguridad de las transacciones agrícolas. Se pide diseñar la solución basada en DDD y microservicios, priorizando trazabilidad, seguridad y escalabilidad.

Escenarios

- Gestión de Productores y Fincas: Registro de productores, ubicación y tipo de cultivo.
- Inventario Agrícola: Control de disponibilidad, precios y calidad del producto.
- Contratos y Pedidos Acuerdos digitales entre productor y comprador.
- Seguimiento del estado del pedido.
- Logística y Transporte: Asignación de transporte y seguimiento de entregas.
- Pagos y Certificaciones: Procesamiento de pagos seguros, Validación de certificaciones de calidad.

II. Se deben tener en cuenta los siguientes aspectos.

- Identifique los subdominios y bounded contexts.
- Modele el dominio (entidades, objetos de valor, agregados y servicios de dominio).
- Proponga la arquitectura de microservicios alineada al dominio.
- Defina APIs y eventos de dominio para integración entre contextos.
- Genere diagramas (Context Map, Casos de Uso, Diagrama de Clases/Agregados, Secuencia).
- Explique decisiones sobre persistencia, resiliencia y observabilidad.

Análisis de subdominios y bounded contexts

Identificación de subdominios.

Siguiendo la metodología de Domain-Driven Design (DDD), clasificamos el dominio en tres categorías según su valor estratégico para el negocio:

CORE DOMAIN: Comercialización.

La comercialización es el Core Domain de nuestra plataforma representa la ventaja competitiva clave del proyecto, ya que constituye la solución innovadora al desafío central identificado por el ministerio. Este dominio se enfoca en conectar de manera eficiente y directa a los productores agrícolas con los compradores potenciales, eliminando intermediarios innecesarios y creando un ecosistema digital transparente. La funcionalidad principal radica en la capacidad de generar acuerdos comerciales digitales completamente trazables, permitiendo un seguimiento detallado desde la negociación inicial hasta la concreción de la transacción. Esta trazabilidad no solo brinda confianza y seguridad jurídica a ambas partes, sino que también proporciona al ministerio datos valiosos para la toma de decisiones en políticas públicas agrícolas. Es precisamente este valor diferencial el que resuelve el problema fundamental del ministerio: la falta de visibilidad y control en las transacciones del sector agrícola, transformando un proceso tradicionalmente opaco en uno digital, eficiente y completamente auditable.

Supporting subdomains:

1. Gestión de productores y fincas

El subdominio de Gestión de Productores y Fincas constituye un componente esencial del Supporting Domain, ya que proporciona la infraestructura necesaria para que el Core Domain pueda operar efectivamente. Aunque esta funcionalidad es necesaria para el funcionamiento de la plataforma, no representa un elemento diferenciador, dado que prácticamente toda plataforma de comercio o Marketplace requiere capacidades similares para gestionar sus usuarios y vendedores. La complejidad de este subdominio es moderada, centrándose en operaciones CRUD (Crear, Leer, Actualizar, Eliminar), validaciones de datos y el manejo de relaciones uno a muchos entre productores y sus respectivas fincas.

2. Inventario agrícola

El subdominio de Inventario Agrícola se clasifica como Supporting Domain debido a que, aunque resulta fundamental para las operaciones diarias de la plataforma, el control de stock es un problema ampliamente resuelto en el ámbito del comercio electrónico y no representa una fuente de innovación o diferenciación competitiva para el proyecto. Las reglas de negocio asociadas al manejo de inventario son relativamente estándar y predecibles: registro de productos disponibles, actualización de cantidades, control de entradas y salidas, y gestión de disponibilidad en tiempo real.

3. Logística y transporte

El subdominio de Logística y Transporte se clasifica como Supporting Domain porque, aunque es indispensable para materializar las transacciones comerciales, la logística actúa como facilitadora de la transacción y no constituye el fin último del proyecto. Es fundamental comprender que el Ministerio no está creando una empresa de transporte ni busca competir en el sector logístico; más bien, este componente representa un medio necesario para completar exitosamente el ciclo de venta iniciado en el Core Domain. La logística permite que los productos negociados lleguen desde las fincas de los productores hasta los compradores, garantizando la entrega física de lo acordado digitalmente, pero no aporta el valor diferenciador estratégico que caracteriza al dominio central.

Generic subdomains:

1. Pagos

El subdominio de Pagos se clasifica como Generic Domain debido a que representa un problema completamente resuelto en la industria tecnológica, sin requerir desarrollo personalizado que aporte valor diferenciador al proyecto. Existen en el mercado pasarelas de pago maduras y ampliamente probadas como Stripe, PayU, MercadoPago y Wompi, que ofrecen soluciones integrales y confiables para el procesamiento de transacciones financieras. Estas plataformas ya cumplen rigurosamente con todas las regulaciones y estándares de seguridad requeridos, como el PCI-DSS (Payment Card Industry Data Security Standard), eliminando la carga de certificaciones complejas y costosas que implicaría desarrollar un sistema de pagos propio.

2. Certificaciones

El subdominio de Certificaciones se clasifica como Generic Domain porque fundamentalmente se trata de un proceso de validación y verificación documental, una funcionalidad común que no aporta diferenciación competitiva al proyecto del ministerio. Este tipo de necesidad es compartida por numerosas plataformas y sistemas en diversos sectores, desde marketplaces hasta portales gubernamentales, donde la verificación de credenciales, licencias o certificados es un requisito operativo estándar pero no estratégico. En el mercado ya existen servicios especializados de verificación de documentos y validación de identidad que pueden integrarse fácilmente mediante APIs, ofreciendo capacidades como OCR (reconocimiento óptico de caracteres), validación de autenticidad, verificación contra bases de datos oficiales y almacenamiento seguro de documentación.

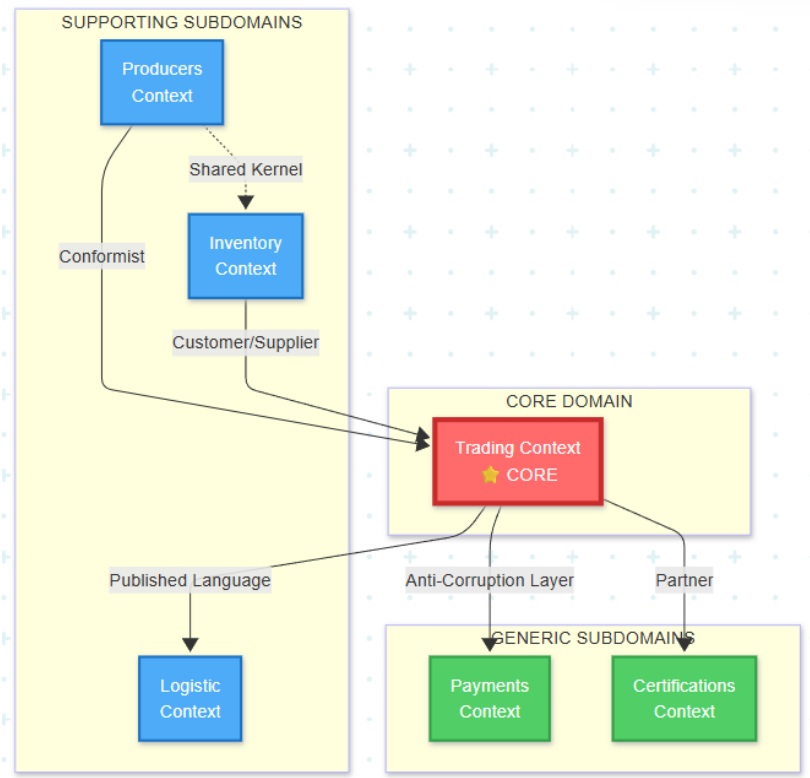
3. Autenticación

El subdominio de Autenticación y Autorización es clasificado como completamente Generic Domain, ya que representa una funcionalidad universalmente necesaria, pero sin ningún componente diferenciador que justifique desarrollo personalizado. Afortunadamente, existen múltiples soluciones maduras y probadas en el mercado que cubren exhaustivamente estos requisitos, tales como Auth0 (solución SaaS empresarial), AWS Cognito (integrado con el ecosistema Amazon), Keycloak (alternativa robusta de código abierto), Firebase Auth (de Google, ideal para desarrollo ágil) y Okta (líder en gestión de identidades).

Identificación de Bounded Contexts

Identificador	Bounded Context	Tipo de subdominio	Complejidad
BC1	Productores (Producers)	Supporting	Media
BC2	Inventario (Inventory)	Supporting	Media-Alta
BC3	Comercialización (Trading)	CORE	Alta
BC4	Logística (Logistic)	Supporting	Media-Alta
BC5	Pagos (Payments)	Generic	Baja
BC6	Certificaciones (Certifications)	Generic	Baja

Context Map – Relaciones entre contextos



1. Productores ── [Conformist]──> Trading

- Productores acepta el modelo de Trading
- Trading es el contexto dominante
- Productores se adapta a las necesidades de Trading

2. Productores ── [Shared Kernel]──> Inventario

- Comparten conceptos básicos de producto y finca
- Pequeño modelo común entre ambos
- Requiere coordinación entre equipos

3. Inventario ── [Customer/Supplier]──> Trading

- Trading es el cliente (Customer)
- Inventario es el proveedor (Supplier)
- Trading consume servicios de Inventario

4. Trading ── [Published Language]──> Logística

- Trading publica eventos estándar
- Logística se suscribe a estos eventos
- Contrato de integración bien definido

5. Trading ── [Anti-Corruption Layer]──> Pagos

- Pagos tiene ACL para pasarelas externas
- Protege el modelo interno de cambios externos
- Traduce conceptos entre dominios

6. Trading ── [Partner]──> Certificaciones

- Colaboración en igualdad
- Ambos se necesitan mutuamente
- Relación bidireccional

Lenguaje ubicuo por contexto

Contexto: Productores

Término	Definición
Productor	Persona natural o jurídica que cultiva productos agrícolas y los ofrece en la plataforma
Finca	Predio rural donde se realiza la actividad agrícola
Cultivo	Clasificación del producto agrícola principal (café, cacao, plátano, etc.)
Estado del productor	Activo e Inactivo
Verificación	Proceso de validación de identidad y datos por el Ministerio
Hectáreas cultivadas	Superficie de tierra dedicada al cultivo

Contexto: Inventario

Término	Definición
Producto	Ítem específico que un productor pone a disposición para la venta
Disponibilidad	Cantidad real de producto que el productor tiene y puede vender
Reserva	Cantidad apartada temporalmente para un contrato en negociación

Stock	Disponibilidad de productos
Precio	Precio sugerido por el productor (puede ser negociado)
Calidad	Grado o categoría del producto (primera, segunda, exportación)
Producto Perecedero	Indica si el producto tiene vida útil limitada

Contexto: Comercialización (Core)

Término	**Definición**
Contrato	Acuerdo digital entre productor y comprador con términos específicos
Pedido	Solicitud concreta de entrega dentro de un contrato
Oferta	Propuesta inicial de términos comerciales
Negociación	Proceso de ajuste de términos antes de firmar contrato
Comprador	Entidad que adquiere productos agrícolas
Estado del contrato	Activo, Completado, Cancelado
Estado del pedido	Pendiente, EnTránsito, Entregado
Términos Comerciales	Condiciones de precio, entrega, pago y penalizaciones

Contexto: logística

Término	**Definición**
Envío	Acuerdo digital entre productor y comprador con términos específicos
Transportista	Solicitud concreta de entrega dentro de un contrato
Ruta	Propuesta inicial de términos comerciales
Vehículo	Proceso de ajuste de términos antes de firmar contrato
Trazabilidad	Entidad que adquiere productos agrícolas
Punto de Control	Registro de ubicación y estado en un momento específico
Incidente	Evento no planificado durante el transporte

Modelado del dominio

BC1: Productores (Producers Context)

Reglas:

1. Un productor debe tener al menos un método de contacto válido
   → Email O Teléfono debe estar presente
2. Una finca debe tener ubicación geográfica definida
   → Latitud Y Longitud son obligatorias
3. Solo productores ACTIVOS pueden crear nuevas publicaciones
   → Estado == "Activo" para publicar inventario
4. Un productor puede tener máximo 50 fincas registradas
   → count(fincas) <= 50
5. El número de documento debe ser único en el sistema
   → No pueden existir dos productores con el mismo documento

Entidades y agregados:

Productor (Aggregate Root - Entity)

Atributos:

- ProductorId: Guid
- NumeroDocumento: string
- TipoDocumento: enum (CC, NIT, CE)
- NombreCompleto: string
- TipoPersona: enum (Natural, Jurídica)
- Contacto: Contacto (VO)
- DireccionResidencia: string
- FechaRegistro: DateTime
- Estado: EstadoProductor (Pendiente, Activo, Suspendido, Inactivo)
- FechaVerificacion: DateTime?
- VerificadoPor: string?
- Fincas: List<Finca>

Comportamientos:

+ AgregarFinca(finca): void
+ ActualizarContacto(contacto): void
+ Activar(verificadorId): void
+ Suspender(motivo): void
+ ValidarParaPublicar(): bool

Finca (Entity dentro del agregado)

Atributos:

- FincaId: Guid
- Nombre: string
- Ubicacion: Ubicacion (VO)
- HectareasCultivadas: decimal
- TipoCultivoPrincipal: TipoCultivo (enum)
- FechaRegistro: DateTime

- EstadoFinca: enum (Activa, Inactiva)

- TieneCertificacionOrganica: bool

Objetos de valor:

Contacto

- Email: string

- Telefono: string

- TelefonoAlternativo: string?

Métodos:

+ ValidarFormato(): bool

+ TieneAlMenosUnMetodo(): bool

Ubicacion

- Latitud: decimal

- Longitud: decimal

- Departamento: string

- Municipio: string

- Vereda: string?

- IndicacionesAcceso: string?

Métodos:

+ CalcularDistancia(otra): double

+ EstaEnColombia(): bool

BC2: Inventario (Inventory Context)

Reglas:

1. La disponibilidad nunca puede ser negativa
   → Disponibilidad >= 0
2. El precio base debe ser mayor a cero
   → PrecioBase.Cantidad >
3. Un producto debe tener unidad de medida definida
   → UnidadMedida != null
4. Productos perecederos deben tener fecha de cosecha
   → Si EsPerecedero entonces FechaCosecha != null
5. El stock efectivo no puede ser negativo
   → (Disponibilidad - Reservas) >= 0

Entidades y agregados:

ProductoAgricola (Aggregate Root - Entity)

Atributos:

- ProductoId: Guid

- ProductorId: Guid (referencia externa)

- FincaId: Guid

- TipoProducto: TipoProducto (enum: Cafe, Cacao, Platano, Aguacate, etc.)

- Descripcion: string

- Disponibilidad: Cantidad (VO)

- Reservas: Cantidad (VO)

- PrecioBase: Money (VO)

- CalidadProducto: Calidad (VO)

- EsPerecedero: bool

- FechaCosecha: DateTime?

- FechaPublicacion: DateTime

- VentanaDisponibilidad: Periodo? (VO)

- Estado: EstadoProducto (Borrador, Activo, Agotado, Despublicado)

- ImagenUrl: string?

Comportamientos:

+ Publicar(): void

+ ActualizarDisponibilidad(nueva): void

+ ReservarInventario(cantidad, reservaId): Result

+ LiberarReserva(cantidad): void

+ ConfirmarVenta(cantidad): void

+ ActualizarPrecio(nuevoPrecio): void

+ ObtenerStockEfectivo(): Cantidad

+ EstaDisponiblePara(fecha): bool

+ Despublicar(motivo): void

Objetos de valor:

Cantidad

- Valor: decimal

- Unidad: UnidadMedida (enum: Kilogramo, Tonelada, Bulto, Arroba, Carga)

Métodos:

+ Sumar(otra): Cantidad

+ Restar(otra): Cantidad

+ ConvertirA(nuevaUnidad): Cantidad

+ EsMayorQue(otra): bool

+ EsSuficientePara(requerida): bool

+ Multiplicar(factor): Cantidad

Money

- Cantidad: decimal

- Moneda: string (COP, USD)

Métodos:

+ Sumar(otro): Money

+ Multiplicar(factor): Money

+ EsMismaMoneda(otro): bool

+ ConvertirA(moneda, tasa): Money

Calidad

- Grado: GradoCalidad (A, B, C, Exportacion)

- TieneCertificacionOrganica: bool

- Descripcion: string

Métodos:

+ EsCalidadExportacion(): bool

Periodo

- FechaInicio: DateTime

- FechaFin: DateTime

Métodos:

+ Contiene(fecha): bool

+ Duracion(): TimeSpan

+ SeSuperpOneCon(otro): bool

Servicios del dominio:

ServicioDisponibilidad

{

+ VerificarDisponibilidad(ProductoId, cantidadSolicitada): bool

+ ReservarInventario(ProductoId, cantidad, reservaId): Result

+ LiberarReservasExpiradas(): void

+ CalcularStockEfectivo(ProductoId): Cantidad

}

ServicioConsultaInventario

{

+ BuscarProductos(TipoProducto?, Calidad?, RangoPrecio?): List<ProductoAgricola>

+ ObtenerDisponibilidadPorZona(Departamento): Dictionary<TipoProducto, Cantidad>

}

BC3: Comercialización (Trading Context) – CORE DOMAIN

Reglas:

1. Un contrato debe tener productor y comprador diferentes
   → ProductorId != CompradorId
2. La cantidad del contrato no puede exceder la disponibilidad
   → CantidadContrato <= InventarioDisponible (verificado en Inventory)
3. El precio acordado debe ser positivo
   → PrecioAcordado.Cantidad > 0
4. Solo contratos activos pueden tener pedidos
   → Estado == "Activo" para crear pedidos
5. La suma de cantidades de pedidos no puede exceder el contrato
   → SUM(Pedidos.Cantidad) <= Contrato.Cantidad
6. Un contrato debe tener fecha de vencimiento posterior a creación
   → FechaVencimiento > FechaCreacion
7. No se puede cancelar un contrato con pedidos entregados
   → Si existe Pedido.Estado == "Entregado" entonces NO cancelar

Entidades y agregados:

Contrato (Aggregate Root - Entity)

Atributos:

- ContratoId: Guid

- ProductorId: Guid (referencia)

- CompradorId: Guid (referencia)

- ProductoId: Guid (referencia)

- TipoProducto: string

- Cantidad: Cantidad (VO)

- PrecioAcordado: Money (VO)

- TerminosEntrega: TerminosComerciales (VO)

- FechaCreacion: DateTime

- FechaVencimiento: DateTime

- Estado: EstadoContrato (Propuesta, Activo, Completado, Cancelado, Vencido)

- MotivoCancelacion: string?

- Pedidos: List<Pedido>

- HistorialEstados: List<CambioEstado>

Comportamientos:

+ CrearPropuesta(): void

+ Aceptar(): void

+ Rechazar(motivo): void

+ Cancelar(motivo): Result

+ CrearPedido(cantidad, direccion): Pedido

+ ActualizarEstadoPedido(pedidoId, nuevoEstado): void

+ Completar(): void

+ Vencer(): void

+ PuedeCrearPedidos(): bool

+ ObtenerCantidadPendiente(): Cantidad

+ CalcularPenalizacion(tipo): Money

Pedido (Entity dentro del agregado Contrato)

Atributos:

- PedidoId: Guid

- ContratoId: Guid

- Cantidad: Cantidad (VO)

- FechaCreacion: DateTime

- FechaEntregaEstimada: DateTime

- FechaEntregaReal: DateTime?

- DireccionEntrega: DireccionEntrega (VO)

- Estado: EstadoPedido (Pendiente, Preparando, EnTransito, Entregado, Cancelado)

- EnvioId: Guid? (referencia a Logistics)

- HistorialEstados: List<CambioEstadoPedido>

- Observaciones: string?

Comportamientos:

+ Preparar(): void

+ IniciarTransporte(envioId): void

+ Entregar(fecha): void

+ Cancelar(motivo): void

+ EstaRetrasado(): bool

+ CalcularDiasRetraso(): int

CambioEstado (Entity)

Atributos:

- FechaCambio: DateTime

- EstadoAnterior: EstadoContrato

- EstadoNuevo: EstadoContrato

- UsuarioResponsable: string

- Comentario: string?

Objetos de valor:

TerminosComerciales (Value Object)

- PlazoEntregaDias: int

- CondicionesPago: CondicionesPago (enum: Anticipo, ContraEntrega, Credito)

- PorcentajeAnticipo: decimal?

- PlazoCredito: int?

- PenalizacionRetraso: Money

- BonificacionAdelanto: Money?

- RequiereCertificacion: bool

- TipoCertificacionRequerida: string?

Métodos:

+ CalcularFechaEntregaMaxima(fechaInicio): DateTime

+ CalcularMontoPenalizacion(diasRetraso): Money

+ ValidarTerminos(): bool

DireccionEntrega (Value Object)

- Direccion: string
- Ciudad: string
- Departamento: string
- CodigoPostal: string?
- Coordenadas: UbicacionVO?
- InstruccionesEspeciales: string?
- PersonaContacto: string
- TelefonoContacto: string

Métodos:

+ FormatearDireccion(): string
+ TieneCoordenadasGPS(): bool

Servicios del dominio:

ServicioNegociacion

{

+ CrearOferta(productorId, compradorId, detalles): ContratoId
+ NegociarTerminos(contratoId, nuevosTerminos): Result
+ AceptarContrato(contratoId, aceptadoPor): Result
+ RechazarContrato(contratoId, motivo): Result
+ ValidarCondicionesMinimas(contrato): bool

}

ServicioPedidos

{

+ CrearPedido(contratoId, cantidad, direccion): PedidoId
+ ActualizarEstadoPedido(pedidoId, nuevoEstado): Result
+ VerificarRetrasos(): List<Pedido>
+ CalcularPenalizaciones(pedidoId): Money
+ NotificarCambioEstado(pedidoId): void }

ServicioValidacionContratos

{

+ ValidarDisponibilidadInventario(contratoId): Result

+ ValidarCertificaciones(contratoId): Result

+ ValidarCapacidadProductor(productorId, cantidad): Result

}

BC4: Logistica (Logistics Context)

Reglas:

1. Un envío debe tener origen y destino diferentes
   → Origen.Ubicacion != Destino.Ubicacion
2. La fecha de salida no puede ser posterior a la entrega estimada
   → FechaSalida <= FechaEntregaEstimada
3. Un vehículo no puede tener dos envíos activos simultáneos
   → Para cada VehiculoId, contar(Envios con Estado "EnRuta") <= 1
4. La capacidad del vehículo debe ser suficiente
   → Cantidad <= CapacidadVehiculo
5. Un envío entregado no puede cambiar de estado
   → Si Estado == "Entregado" entonces no permitir cambios
6. Debe haber al menos un punto de control en estado "EnRuta"
   → Si Estado == "EnRuta" entonces count(PuntosControl) >= 1

Entidades y agregados:

Envio (Aggregate Root - Entity)

Atributos:

- EnvioId: Guid

- PedidoId: Guid (referencia a Trading)

- TransportistaId: Guid

- VehiculoId: Guid

- Origen: Ubicacion (VO)

- Destino: Ubicacion (VO)

- Cantidad: Cantidad (VO)

- RequiereRefrigeracion: bool

- TemperaturaRequerida: decimal?

- FechaSalida: DateTime

- FechaEntregaEstimada: DateTime

- FechaEntregaReal: DateTime?

- Estado: EstadoEnvio (Programado, EnRuta, Entregado, Cancelado, Incidente)

- PuntosControl: List<PuntoControl>

- Incidentes: List<Incidente>

- RutaPlanificada: Ruta (VO)

- FirmaEntrega: FirmaDigital (VO)?

- FotosEntrega: List<string>?

Comportamientos:

+ Programar(): void

+ IniciarRuta(): void

+ RegistrarPuntoControl(ubicacion, observaciones): void

+ ReportarIncidente(tipo, descripcion, ubicacion): void

+ ResolverIncidente(incidenteId, resolucion): void

+ Entregar(firma, fotos): void

+ Cancelar(motivo): void

+ EstaEnRuta(): bool

+ CalcularDistanciaRecorrida(): decimal

+ CalcularTiempoEstimadoLlegada(): TimeSpan

+ TieneRetrasoSignificativo(): bool

PuntoControl (Entity)

Atributos:

- PuntoControlId: Guid

- UbicacionActual: Ubicacion (VO)

- FechaHora: DateTime

- Temperatura: decimal?

- Observaciones: string?

- FotoOpcional: string?

Comportamientos:

+ EstaEnRuta(rutaPlanificada): bool

+ CalcularDesviacion(rutaPlanificada): decimal

Incidente (Entity)

Atributos:

- IncidenteId: Guid

- TipoIncidente: TipoIncidente (enum: Accidente, DesviRuta, FallaVehiculo, etc.)

- Descripcion: string

- UbicacionIncidente: Ubicacion (VO)

- FechaReporte: DateTime

- Estado: EstadoIncidente (Reportado, EnAtencion, Resuelto)

- Resolucion: string?

- FechaResolucion: DateTime?

- ImpactoEntrega: bool

Comportamientos:

+ Resolver(resolucion): void

+ EsCritico(): bool

Objetos de valor:

Ruta

- Puntos: List<Ubicacion>

- DistanciaEstimada: decimal (en km)

- TiempoEstimado: TimeSpan

Métodos:

+ CalcularDistanciaTotal(): decimal

+ ObtenerPuntoMasCercano(ubicacionActual): Ubicacion

+ EstaEnRuta(ubicacion, tolerancia): bool

FirmaDigital

- FirmaBase64: string

- NombreFirmante: string

- DocumentoFirmante: string

- FechaHora: DateTime

Métodos:

+ ValidarFirma(): bool

Servicios del dominio:

ServicioAsignacionTransporte

{

+ AsignarTransportista(envioId, criterios): TransportistaId

+ SeleccionarVehiculo(cantidad, requiereRefrigeracion): VehiculoId

+ CalcularRutaOptima(origen, destino): Ruta

+ EstimarTiempoViaje(ruta, condiciones): TimeSpan

}

ServicioSeguimiento

{

+ RegistrarPuntoControl(envioId, ubicacion, temperatura): void

+ VerificarDesvioRuta(envioId, ubicacionActual): bool

+ CalcularETA(envioId): DateTime

+ NotificarCambioEstado(envioId, nuevoEstado): void

}

ServicioIncidentes

{

+ ReportarIncidente(envioId, tipo, descripcion, ubicacion): IncidenteId

+ EvaluarImpacto(incidente): ImpactoIncidente

+ NotificarIncidente(incidenteId, destinatarios): void

}

BC5: Pagos (Payments Context)

Reglas:

1. El monto total debe ser positivo
   → MontoTotal > 0
2. La suma de distribuciones debe igualar el monto total
   → SUM(Distribuciones.Monto) == MontoTotal
3. Una transacción completada no puede cambiar de estado
   → Si Estado == "Completado" entonces es inmutabl
4. Debe haber al menos un beneficiario
   → count(Beneficiarios) >= 1
5. El porcentaje de comisión debe estar entre 0 y 20%
   → 0 <= PorcentajeComision <= 20

Entidades y agregados:

Transaccion (Aggregate Root - Entity)

Atributos:

- TransaccionId: Guid

- ContratoId: Guid (referencia)

- PedidoId: Guid? (referencia)

- TipoTransaccion: TipoPago (Anticipo, PagoEntrega, PagoFinal)

- MontoTotal: Money (VO)

- MetodoPago: MetodoPago (enum: TarjetaCredito, TarjetaDebito, PSE, etc.)

- Estado: EstadoTransaccion (Pendiente, Procesando, Completado, Fallido, Reembolsado)

- FechaCreacion: DateTime

- FechaCompletacion: DateTime?

- ReferenciaPasarela: string?

- Beneficiarios: List<Beneficiario>

- IntentosReintento: int

- ErrorMensaje: string?

Comportamientos:

+ Iniciar(): void

+ ProcesarCallback(resultado): void

+ Completar(referencia): void

+ Fallar(error): void

+ Reintentar(): void

+ Reembolsar(motivo): void

+ CalcularDistribucion(): List<Distribucion>

Beneficiario (Entity)

Atributos:

- BeneficiarioId: Guid

- TipoBeneficiario: TipoBeneficiario (Productor, Transportista, Plataforma)

- UsuarioId: Guid

- Porcentaje: decimal

- MontoCalculado: Money (VO)

- CuentaBancaria: CuentaBancaria (VO)

- EstadoPago: EstadoPagoBeneficiario (Pendiente, Procesado, Completado, Fallido)

- FechaPago: DateTime?

Comportamientos:

+ CalcularMonto(montoTotal): Money

+ MarcarComoPagado(): void

Objetos de valor:

CuentaBancaria

- Banco: string

- TipoCuenta: TipoCuenta (Ahorros, Corriente)

- NumeroCuenta: string

- TitularCuenta: string

- DocumentoTitular: string

Métodos:

+ Validar(): bool

+ EsCuentaColombia(): bool

Anti-Corruption Layer (ACL):

IPasarelaPagoAdapter

{

+ IniciarPago(transaccion): ReferenciaExterna

+ ConsultarEstado(referencia): EstadoPago

+ ProcesarWebhook(payload): ResultadoWebhook

+ Reembolsar(referencia, monto): Result

+ DistribuirPago(referencia, beneficiarios): Result

}

// **Implementaciones concretas:**

- PayUAdapter

- StripeAdapter

- MercadoPagoAdapter

BC6: Certificaciones (Certifications Context)

Reglas:

1. La fecha de emisión debe ser anterior al vencimiento
   → FechaEmision < FechaVencimiento
2. Una certificación vencida no puede estar vigente
   → Si FechaVencimiento < HOY entonces Estado != "Vigente"
3. Debe tener entidad certificadora
   → EntidadCertificadora != nul
4. El documento debe estar presente para certificaciones no automáticas
   → Si RequiereRevisionManual entonces DocumentoUrl != null

Entidades y agregados:

Certificacion (Aggregate Root - Entity)

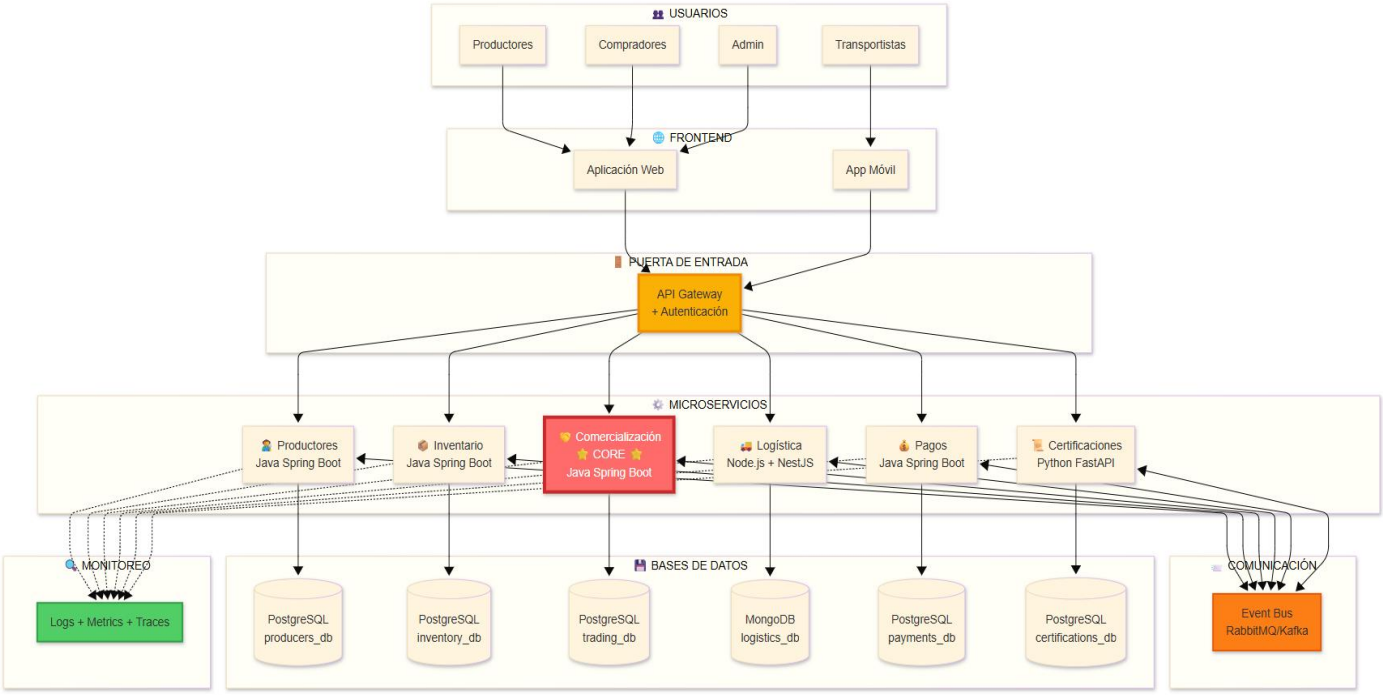
Atributos:

- CertificacionId: Guid
- ProductorId: Guid? (referencia)
- ProductoId: Guid? (referencia)
- FincaId: Guid? (referencia)
- TipoCertificacion: TipoCertificacion (enum: BPA, Organica, GlobalGAP, etc.)
- EntidadCertificadora: string
- NumeroCertificado: string
- FechaEmision: DateTime
- FechaVencimiento: DateTime
- DocumentoUrl: string?
- Estado: EstadoCertificacion (Pendiente, EnValidacion, Vigente, Rechazada, Vencida)
- ObservacionesValidacion: string?
- ValidadoPor: string?
- FechaValidacion: DateTime?

Comportamientos:

+ Registrar(): void
+ IniciarValidacion(validador): void
+ Aprobar(validador, observaciones): void
+ Rechazar(validador, motivo): void
+ Vencer(): void
+ RenovarSolicitud(): Certificacion
+ EstaVigente(): bool

Arquitectura de microservicios



Apis y eventos de dominio

Apis por microservicio

Producers Service API

- POST /api/v1/productores
  Body: { numeroDocumento, tipoDocumento, nombreCompleto, contacto, ... }
  Response: { productorId }

- GET /api/v1/productores/{id}
  Response: { productor con fincas }

- PUT /api/v1/productores/{id}
  Body: { campos a actualizar }
  Response: { success }

- POST /api/v1/productores/{id}/fincas
  Body: { nombre, ubicacion, hectareas, tipoCultivo }
  Response: { fincaId }

- GET /api/v1/productores/{id}/fincas
  Response: [lista de fincas]

- PUT /api/v1/productores/{id}/activar
  Body: { verificadorId }
  Response: { success }

- PUT /api/v1/productores/{id}/suspender
  Body: { motivo }
  Response: { success }

- GET /api/v1/productores?estado={estado}&departamento={depto}
  Response: [lista paginada de productores]

Inventory Service API

- POST /api/v1/productos
  Body: { productorId, fincaId, tipoProducto, disponibilidad, precio, ... }
  Response: { productoId }

- GET /api/v1/productos/{id}
  Response: { producto completo }

- PUT /api/v1/productos/{id}/disponibilidad
  Body: { nuevaDisponibilidad }
  Response: { success }

- GET
  /api/v1/productos/buscar?tipo={tipo}&disponible=true&minPrecio={min}&maxPrecio={max}
  Response: [lista de productos]

- POST /api/v1/productos/{id}/reservar
  Body: { cantidad, reservaId }
  Response: { success, stockEfectivo }

- POST /api/v1/productos/{id}/liberar-reserva
  Body: { reservaId }
  Response: { success }

- GET /api/v1/productos?productorId={id}
  Response: [productos del productor]

Trading Service API (CORE)

- POST /api/v1/contratos
  Body: { productorId, compradorId, productoId, cantidad, precioAcordado, terminos }
  Response: { contratoId }

- GET /api/v1/contratos/{id}
  Response: { contrato con pedidos }

- PUT /api/v1/contratos/{id}/aceptar
  Body: { aceptadoPor }
  Response: { success }

- PUT /api/v1/contratos/{id}/rechazar
  Body: { motivo }
  Response: { success }

- PUT /api/v1/contratos/{id}/cancelar
  Body: { motivo }
  Response: { success }

- POST /api/v1/contratos/{id}/pedidos
  Body: { cantidad, direccionEntrega, fechaEntregaEstimada }
  Response: { pedidoId }

- GET /api/v1/pedidos/{id}
  Response: { pedido con estado y seguimiento }

- PUT /api/v1/pedidos/{id}/estado
  Body: { nuevoEstado, observaciones }
  Response: { success }

- GET /api/v1/contratos?productorId={id}&estado={estado}
  Response: [lista de contratos]

- GET /api/v1/contratos/{id}/historial
  Response: [historial de cambios de estado]

Logistics Service API

- POST /api/v1/envios
  Body: { pedidoId, origen, destino, cantidad, requiereRefrigeracion }
  Response: { envioId }

- GET /api/v1/envios/{id}
  Response: { envío completo }

- PUT /api/v1/envios/{id}/asignar
  Body: { transportistaId, vehiculoId }
  Response: { success }

- PUT /api/v1/envios/{id}/iniciar
  Response: { success, rutaPlanificada }

- POST /api/v1/envios/{id}/ubicacion
  Body: { latitud, longitud, temperatura, observaciones }
  Response: { success }

- GET /api/v1/envios/{id}/seguimiento
  Response: { ubicacionActual, puntosControl, ETA, estado }

- POST /api/v1/envios/{id}/incidentes
  Body: { tipoIncidente, descripcion, ubicacion }
  Response: { incidenteId }

- PUT /api/v1/envios/{id}/entregar
  Body: { firma, fotos[] }
  Response: { success }

- GET /api/v1/envios?estado=EnRuta
  Response: [envíos activos con ubicación]

Payments Service API

- POST /api/v1/transacciones
  Body: { contratoId, pedidoId, monto, tipoTransaccion, metodoPago, beneficiarios[] }
  Response: { transaccionId, urlPago }

- GET /api/v1/transacciones/{id}
  Response: { transacción con estado }

- POST /api/v1/webhooks/pasarela
  Body: { datos de la pasarela }
  Response: { acknowledged }

- POST /api/v1/transacciones/{id}/reintentar
  Response: { success, nuevaUrl? }

- GET /api/v1/transacciones?contratoId={id}
  Response: [transacciones del contrato]

- POST /api/v1/transacciones/{id}/reembolsar
  Body: { motivo }
  Response: { success }

Certifications Service API

- POST /api/v1/certificaciones
  Body: { productorId, tipoCertificacion, entidadCertificadora, documento, fechas }
  Response: { certificacionId }

- GET /api/v1/certificaciones/{id}
  Response: { certificación completa }

- PUT /api/v1/certificaciones/{id}/validar
  Body: { validadorId, observaciones }
  Response: { success }

- PUT /api/v1/certificaciones/{id}/rechazar
  Body: { validadorId, motivo }
  Response: { success }

- GET /api/v1/certificaciones?productorId={id}&tipo={tipo}&estado={estado}
  Response: [certificaciones]

- GET /api/v1/certificaciones/proximas-vencer?dias={n}
  Response: [certificaciones por vencer]

Eventos de Dominio

Producers Context:

- ProductorRegistrado
- FincaAgregada
- ProductorActivado
- ProductorSuspendido
- ContactoActualizado
- FincaActualizada

Inventory Context:

- ProductoPublicado
- InventarioActualizado
- ProductoReservado
- ReservaLiberada
- ProductoVendido
- PrecioActualizado
- ProductoDespublicado
- ReservaExpirada

Trading Context (CORE):

- ContratoCreado
- ContratoAceptado
- ContratoRechazado
- ContratoCancelado
- ContratoCompletado
- ContratoVencido
- PedidoCreado
- EstadoPedidoCambiado
- PedidoEntregado
- PedidoCancelado
- PenalizacionAplicada

Logistics Context:

- EnvioCreado
- EnvioAsignado
- EnvioIniciado
- UbicacionActualizada
- IncidenteReportado
- IncidenteResuelto
- EnvioEntregado
- EnvioCancelado
- AlertaDesvioRuta
- AlertaRetraso

Payments Context:

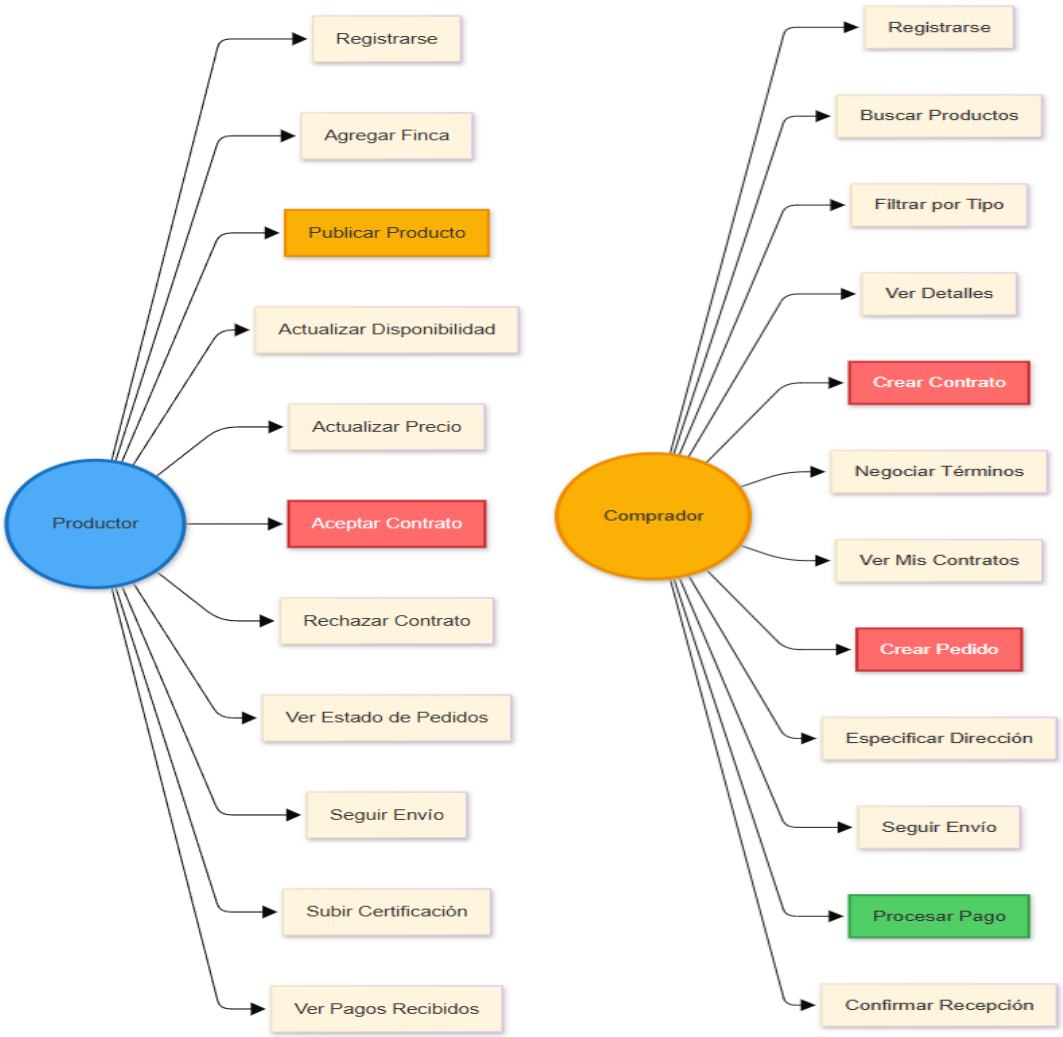
- PagoIniciado
- PagoCompletado
- PagoFallido
- PagoReembolsado
- PagoBeneficiarioCompletado

Certifications Context:

- CertificacionRegistrada
- CertificacionValidada
- CertificacionRechazada
- CertificacionVencida
- AlertaProximoVencimiento

Diagramas

Diagrama de casos de usos



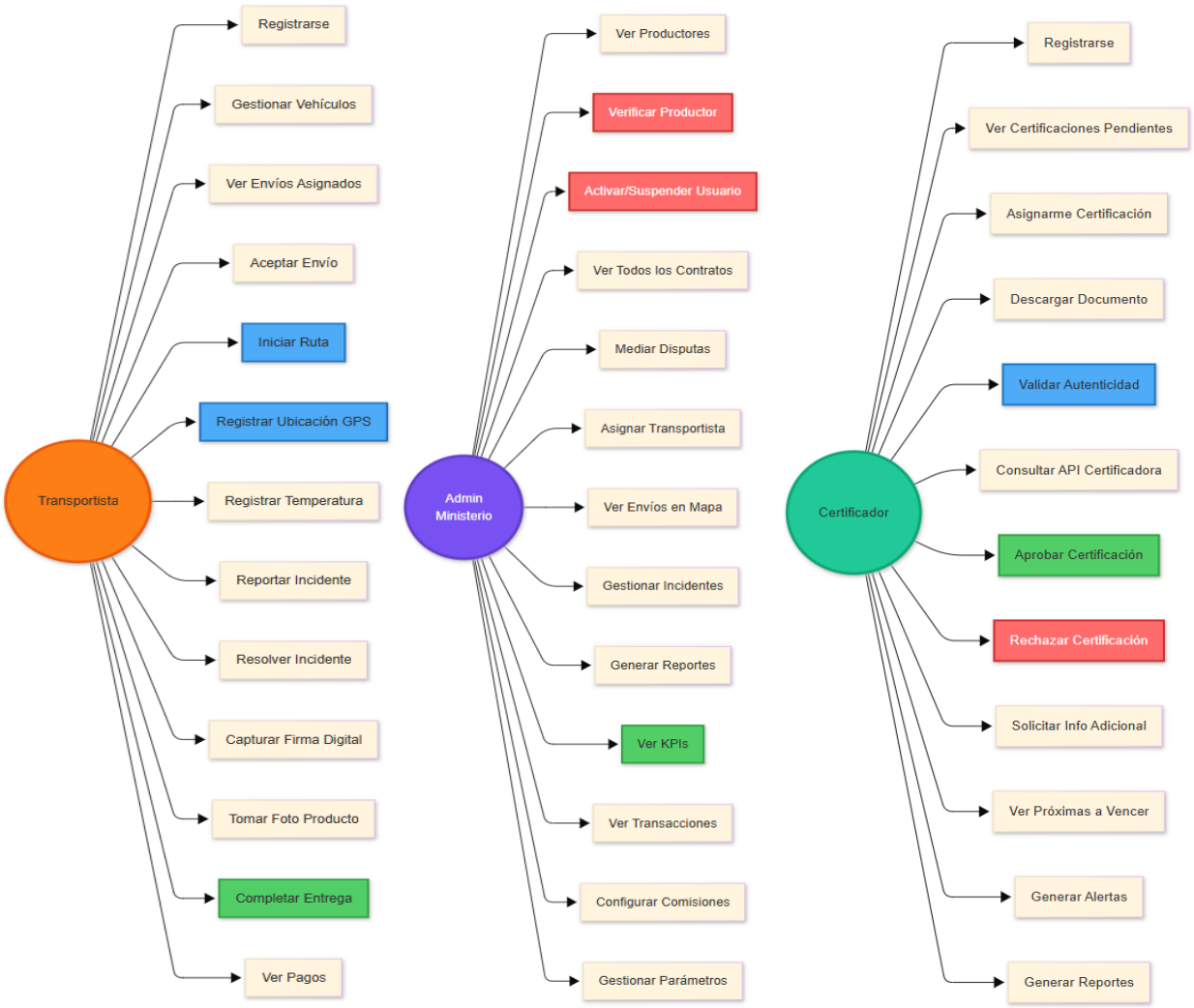

Transportista
Registrarse
Gestionar Vehículos
Ver Envíos Asignados
Aceptar Envío
Iniciar Ruta
Registrar Ubicación GPS
Registrar Temperatura
Reportar Incidente
Resolver Incidente
Capturar Firma Digital
Tomar Foto Producto
Completar Entrega
Ver Pagos
Admin Ministerio
Ver Productores
Verificar Productor
Activar/Suspender Usuario
Ver Todos los Contratos
Mediar Disputas
Asignar Transportista
Ver Envíos en Mapa
Gestionar Incidentes
Generar Reportes
Ver KPIs
Ver Transacciones
Configurar Comisiones
Gestionar Parámetros
Certificador
Registrarse
Ver Certificaciones Pendientes
Asignarme Certificación
Descargar Documento
Validar Autenticidad
Consultar API Certificadora
Aprobar Certificación
Rechazar Certificación
Solicitar Info Adicional
Ver Próximas a Vencer
Generar Alertas
Generar Reportes

Diagrama de clases

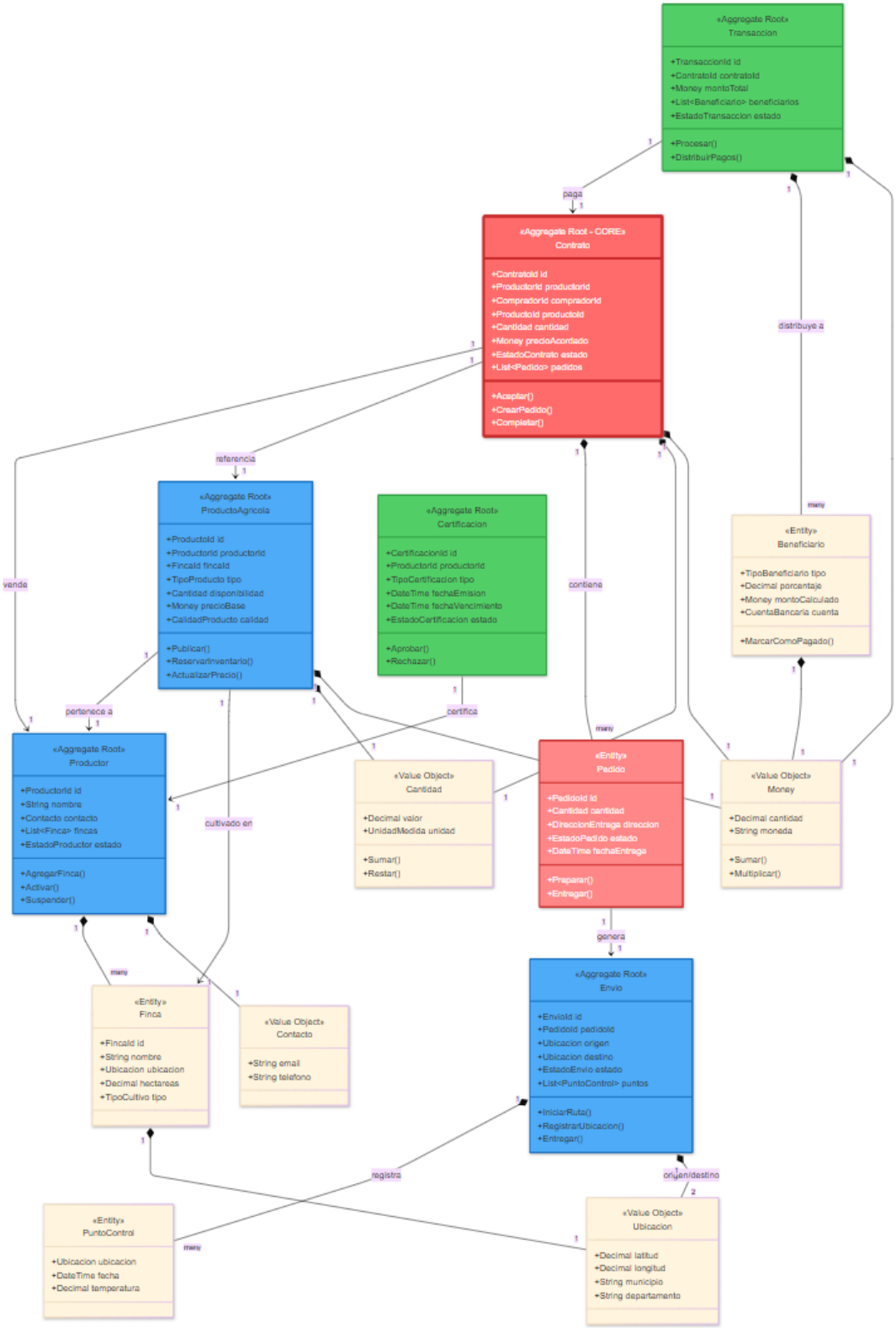
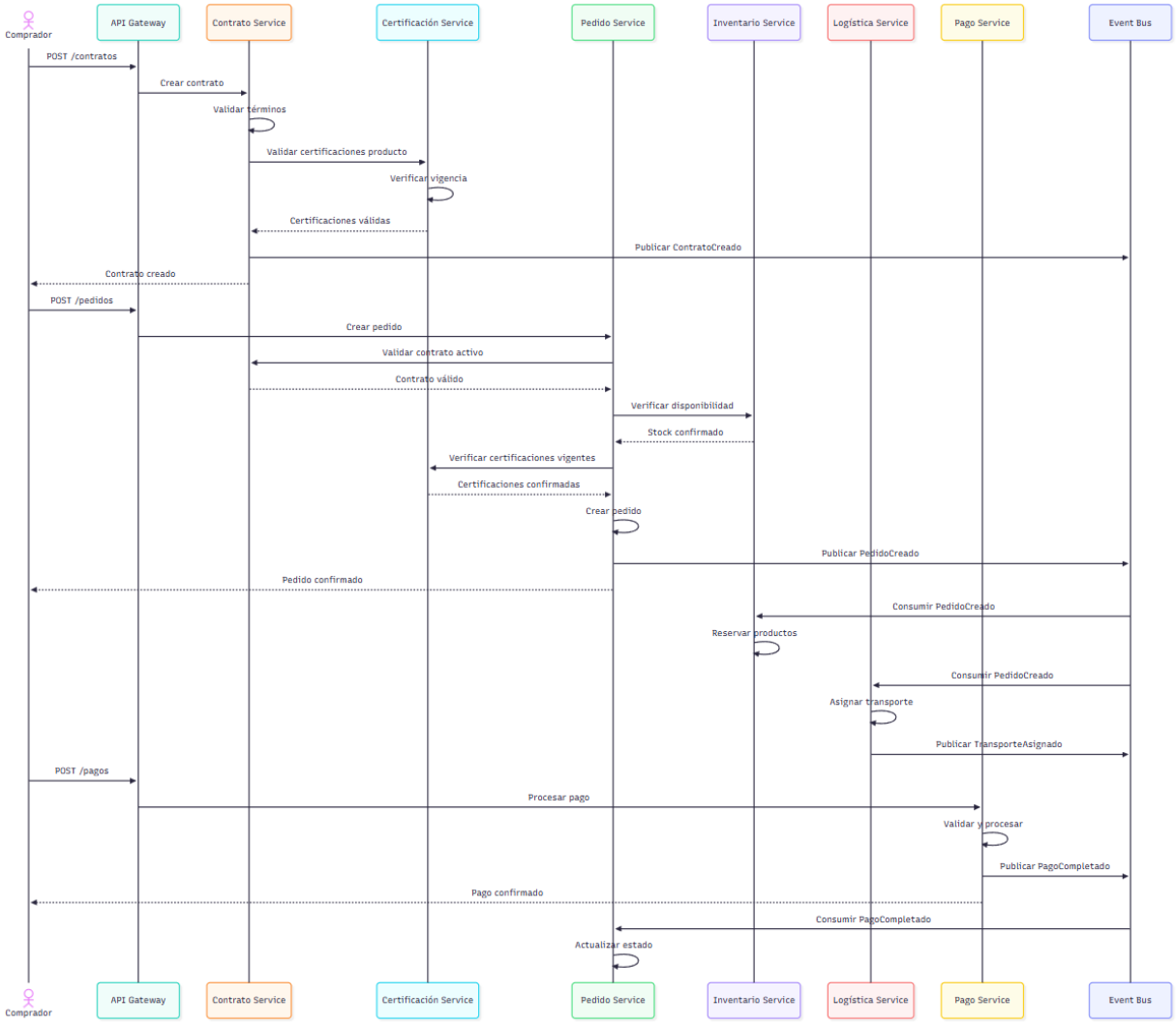


Diagrama de secuencia



Decisiones técnicas

Persistencia

Microservicio	Lenguaje/Framework	Base de Datos	Justificación
Producers	Java Spring Boot	PostgreSQL 15	Datos relacionales estructurados, relación 1:N (Productor-Fincas)
Inventory	Java Spring Boot	PostgreSQL 15	**Transacciones ACID críticas** para reservas de inventario
Trading (CORE)	Java Spring Boot	PostgreSQL 15	**El más crítico:** Integridad referencial, auditoría completa, transacciones complejas
Logistics	Node.js + NestJS	MongoDB 6.0	**Datos geoespaciales** (índices 2dsphere), alta frecuencia de escritura (GPS cada 5-10 min)
Payments	Java Spring Boot	PostgreSQL 15	**Consistencia financiera crítica**, compliance, auditoría
Certifications	Python FastAPI	PostgreSQL 15 + S3	Metadatos en PostgreSQL, documentos PDF/imágenes en S3

Resiliencia

Aspecto	Tecnología	Justificación
Circuit Breaker	Resilience4j	**Propósito:** Evitar fallos en cascada cuando un servicio externo falla.
Retry Pattern	Resilience4j	**Propósito:** Reintentar operaciones que pueden fallar temporalmente.
Timeout Pattern	Resilience4j	**Propósito:** Evitar esperas infinitas poniendo liites de tiempo por operación.
Bulkhead Pattern	Resilience4j	**Propósito:** Aislar recursos para que un servicio lento no afecte todo. **Propósito:** Aislar recursos para que un servicio lento no afecte todo.
Saga Pattern	Event Bus + Compensaciones	**Propósito:** Manejar transacciones distribuidas sin transacciones ACID entre servicios.
Health Checks	Spring Boot Actuator + K8s	**Propósito:** Identificar el estado de los servicios

Observabilidad

Aspecto	**Tecnología**
Logs	- **Elasticsearch:** Almacenamiento y búsqueda - **Logstash / Fluentd:** Agregación - **Kibana:** Visualización
Metrics	- **Prometheus:** Recolección de métricas time-series - **Grafana:** Visualización con dashboards
Tracing	- Jaeger/Zipkin para distributed tracing - Rastrear una request a través de múltiples microservicios.