**Modularización y componetización.**

Elabore para cada uno de los siguientes enunciados:

1. Identifique las funcionalidades (requisitos funcionales)
2. Modularice el sistema que dará solución a las funcionalidades identificando componentes
3. Identifique las interfaces – conectores que permitirán comunicar los componentes
4. Agregue un análisis breve que justifique las decisiones de diseño para el modelo propuesto.
5. Socialice en clase sus propuestas de diseño.

**Ejercicio 1.** Un centro médico requiere un sistema web para que pacientes puedan reservar consultas, médicos gestionen su agenda y los administradores controlen el sistema.

**Desarrollo**

**1. Requerimientos funcionales**

* RF001: El paciente debe poder registrarse e iniciar sesión.
* RF002: El paciente debe poder buscar un médico y reservar una cita.
* RF003: El médico debe poder gestionar su agenda (ver, modificar, cancelar).
* RF004: El administrador debe poder gestionar usuarios y médicos.
* RF005: El sistema debe confirmar y guardar la cita.

**2. Componentes**

* **GestorUsuarios** = para el RF001
* **GestorAgenda** = para el RF003
* **GestorCitas** = para el RF002 y RF005
* **GestorAdmin** = para el RF004

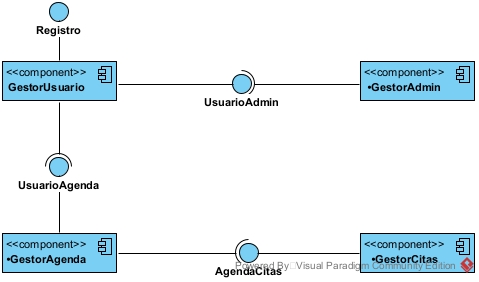
**3. Interfaces de comunicación**

* UsuarioAgenda= GestorUsuarios ↔ GestorAgenda → para validar quién puede ver/modificar horarios.
* AgendaCita = GestorAgenda ↔ GestorCitas → para validar disponibilidad antes de confirmar una cita.
* UsuarioAgenda = GestorAdmin ↔ GestorUsuarios / GestorAgenda → para crear y modificar datos globales.

**4. Justificación del diseño**

El sistema se divide en componentes según las responsabilidades:

* Cada actor (paciente, médico, admin) interactúa con un módulo diferente.
* Esto facilita el mantenimiento, ya que si cambia la lógica de citas no afecta la de usuarios.
* La comunicación entre módulos es simple y clara, reduciendo el acoplamiento.



**Ejercicio 2.** Una startup quiere desarrollar una aplicación móvil y web tipo Rappi o Uber Eats, donde los usuarios pueden pedir comida, los restaurantes gestionan pedidos y los repartidores hacen las entregas.

**Desarrollo**

1. **Requerimientos funcionales**

* RF001: El cliente debe poder registrarse e iniciar sesión.
* RF002: El cliente debe poder buscar restaurantes y productos.
* RF003: El cliente debe poder hacer un pedido y pagarlo en línea.
* RF004: El restaurante debe poder recibir y confirmar pedidos.
* RF005: El repartidor debe poder aceptar y entregar pedidos.
* RF006: El sistema debe permitir seguimiento en tiempo real del pedido.

**2. Componentes**

* GestorUsuarios = RF001
* GestorPedidos = RF003 Y RF004
* GestorRestaurantes = RF002
* GestorPagos = RF003
* GestorEntregas = RF005 Y RF006

**3. Interfaces de comunicación**

* UsuarioPedido = GestorUsuarios ↔ GestorPedidos → valida identidad del cliente al realizar pedidos.
* PedidoRestaurante = GestorPedidos ↔ GestorRestaurantes → envía pedidos y recibe confirmación.
* PedidoPago = GestorPedidos ↔ GestorPagos → procesa el pago antes de confirmar el pedido.
* PedidoEntregas = GestorPedidos ↔ GestorEntregas → asigna pedido a un repartidor.
* EntregasUsuario= GestorEntregas ↔ GestorUsuarios → actualiza al cliente sobre el estado del pedido.

**4. Justificación del diseño**

* Se separan responsabilidades según los actores (clientes, restaurantes, repartidores).
* El GestorPedidos actúa como núcleo que coordina con los demás componentes.
* El GestorPagos se mantiene independiente para permitir integrar distintos métodos de pago.
* Esta organización permite escalar el sistema fácilmente (más usuarios, más pedidos).
* Diagrama

  Descripción generada automáticamente

**Ejercicio 3.** Un colegio tiene un sistema heredado de biblioteca que mezcla lógica de préstamo, usuarios, catálogo y reportes en un solo módulo monolítico. El sistema debe ser **refactorizado y rediseñado** usando principios modernos de diseño.

**Desarrollo**

**1. Requerimientos funcionales**

* RF001: El sistema debe permitir registrar y gestionar usuarios (estudiantes, profesores).
* RF002: El sistema debe permitir consultar el catálogo de libros.
* RF003: El sistema debe permitir realizar préstamos y devoluciones.
* RF004: El sistema debe generar reportes de uso (libros más prestados, usuarios activos, etc.).
* RF005: El sistema debe manejar sanciones en caso de retrasos en devoluciones.

**2. Componentes**

* GestorUsuarios = RF001
* GestorCatalogo = RF002
* GestorPrestamos = RF003 Y RF005
* GestorReportes = RF004 Y RF005

**3. Interfaces de comunicación**

* UsuarioPrestamo = GestorUsuarios ↔ GestorPrestamos → valida al usuario antes de realizar un préstamo.
* CatalogoPrestamo = GestorCatalogo ↔ GestorPrestamos → consulta disponibilidad y actualiza inventario.
* PrestamoReporte = GestorPrestamos ↔ GestorReportes → envía información de uso para generar estadísticas y reportar las sanciones.

**4. Justificación del diseño**

* Se descompone el monolito en módulos independientes para mejorar el mantenimiento y la escalabilidad.
* Cada componente asume una responsabilidad clara (usuarios, catálogo, préstamos, reportes).
* La separación facilita futuras integraciones, como migración a una app web o integración con sistemas de pago de multas.

