**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.

**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.

**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.

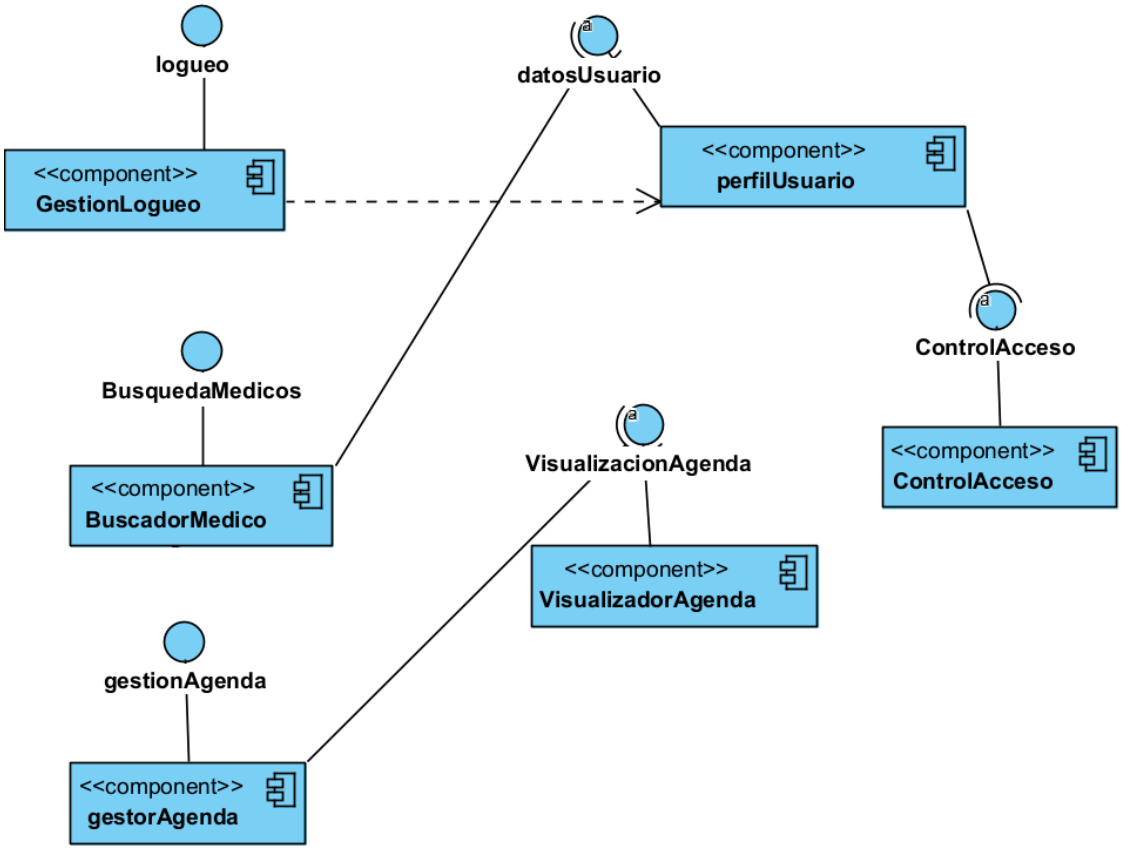
1) Ejercicio 1.

1.1) Requerimientos funcionales:

- RF1: El sistema debe permitir a los pacientes, médicos y administradores iniciar sesión con sus credenciales.
- RF2: El sistema debe validar que las credenciales ingresadas correspondan a un usuario registrado.
- RF3: El sistema debe permitir cerrar sesión de manera segura.
- RF4: El sistema debe permitir a los usuarios (pacientes y médicos) consultar y actualizar su información personal.
- RF5: El sistema debe mostrar al paciente sus datos de contacto, historial de citas y médicos asociados.
- RF6: El sistema debe permitir a los médicos registrar su especialidad, disponibilidad y horarios de atención.
- RF7: El sistema debe definir distintos roles de usuario (paciente, médico, administrador).

- RF8: El sistema debe permitir a los administradores gestionar permisos de acceso a funcionalidades críticas.
- RF9: El sistema debe restringir a los pacientes la gestión de agendas de médicos y permitirles solo reservar citas.
- RF10: El sistema debe permitir a los pacientes buscar médicos por nombre, especialidad o disponibilidad.
- RF11: El sistema debe mostrar los datos básicos del médico seleccionado (nombre, especialidad, horarios disponibles).
- RF15: El sistema debe mostrar a los pacientes las citas programadas, con fecha, hora y médico asignado.
- RF16: El sistema debe mostrar a los médicos su agenda completa de citas con los pacientes registrados.
- RF17: El sistema debe notificar al usuario (paciente/médico) sobre citas nuevas, cancelaciones o reprogramaciones.

2,3)




4 ) Justificacion.

Para empezar, he separado la gestión de logueo y control de acceso en componentes diferentes, lo que me permite reforzar la seguridad y manejar permisos de manera clara para pacientes, médicos y administradores. De esta forma, cada usuario tiene acceso solo a la información y funcionalidades que necesita.

También he creado un perfil de usuario centralizado que mantiene toda la información personal y médica en un solo lugar, lo que simplifica la gestión y evita la duplicidad de datos. Esto me permite tener una visión completa de cada usuario y garantizar que la información sea precisa y actualizada.

En cuanto a la agenda médica, he decidido dividirla en dos componentes: Gestor de Agenda y Visualizador de Agenda. El Gestor de Agenda es para los médicos, que pueden crear y administrar horarios de citas, mientras que el Visualizador de Agenda es para pacientes y médicos, que pueden consultar citas programadas. Esta separación mejora la experiencia del usuario y reduce la complejidad del componente.

Además, he creado un buscador de médicos como componente independiente, lo que garantiza que la búsqueda de profesionales sea ágil y reutilizable. Esto también me permite agregar filtros avanzados en el futuro, como especialidad, disponibilidad inmediata o ubicación.

Finalmente, he diseñado la arquitectura del sistema para atender a múltiples usuarios, incluyendo pacientes, médicos y administradores, sin sobrecargar un único módulo. Esto me permite ofrecer una experiencia personalizada y eficiente para cada tipo de usuario, sin comprometer la seguridad y la escalabilidad del sistema.

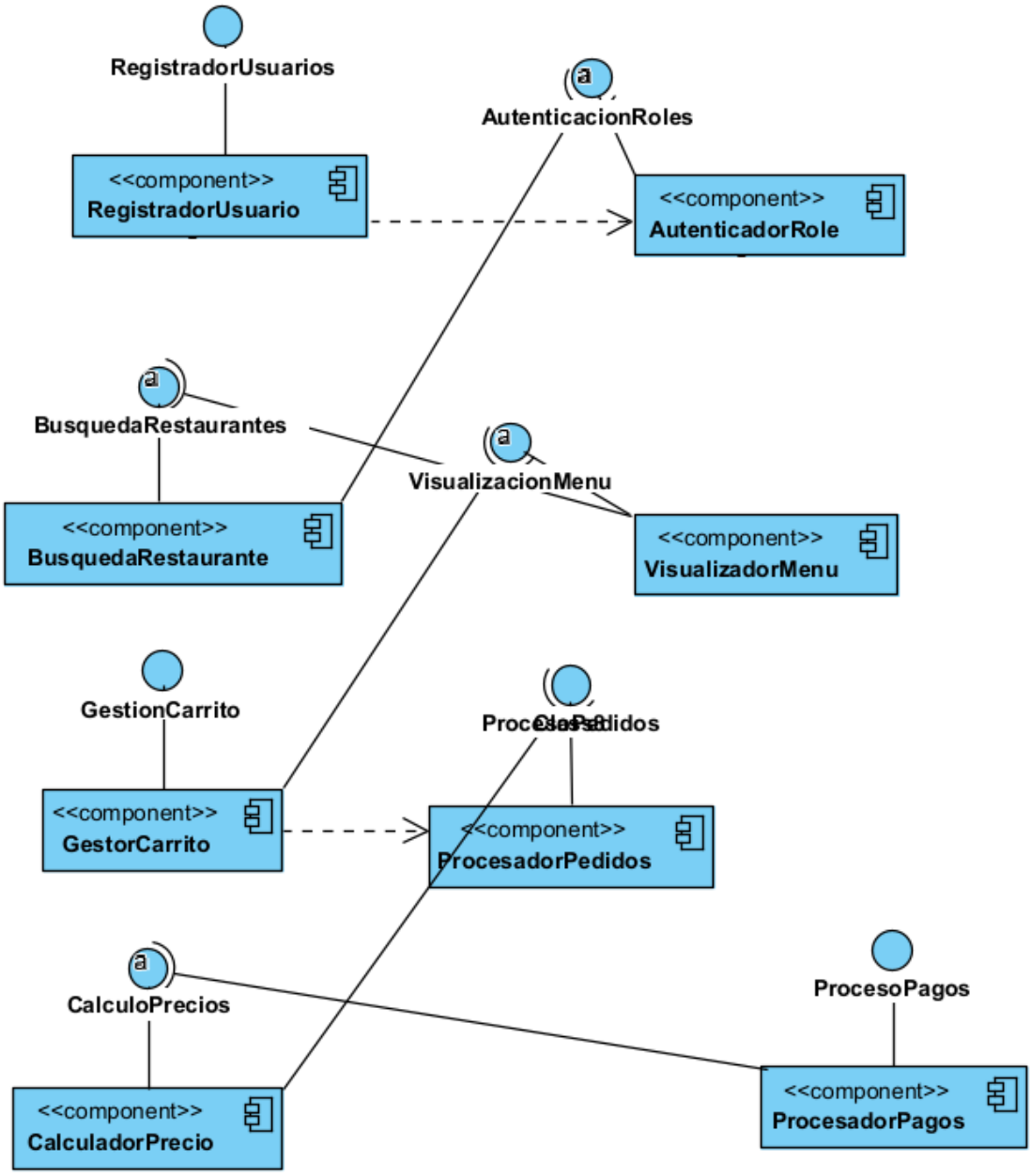
2) Ejercicio 2.

1.1) Requerimientos Funcionales

- RF1: El sistema debe permitir a los usuarios registrarse como cliente, restaurante o repartidor.
- RF2: El sistema debe validar la información ingresada durante el registro (correo, teléfono, contraseña, etc.).
- RF3: El sistema debe permitir la actualización de la información de perfil.
- RF4: El sistema debe autenticar a los usuarios mediante correo/contraseña o redes sociales.
- RF5: El sistema debe gestionar roles diferenciados: cliente, restaurante y repartidor.

- RF6: El sistema debe otorgar accesos de acuerdo al rol (ejemplo: los clientes hacen pedidos, los restaurantes gestionan menús/pedidos, los repartidores gestionan entregas).
- RF7: El sistema debe permitir a los usuarios buscar restaurantes por nombre, categoría de comida, ubicación o calificación.
- RF8: El sistema debe mostrar información básica del restaurante (nombre, dirección, horarios, calificación).
- RF9: El sistema debe mostrar el menú del restaurante seleccionado con precios y descripciones de los productos.
- RF10: El sistema debe permitir la visualización de promociones y combos disponibles.
- RF11: El sistema debe permitir a los usuarios agregar productos al carrito de compras.
- RF12: El sistema debe permitir modificar el carrito (agregar, eliminar o cambiar cantidades de productos).
- RF13: El sistema debe mostrar el total preliminar antes de confirmar el pedido.
- RF14: El sistema debe permitir a los clientes confirmar un pedido desde el carrito.
- RF15: El sistema debe notificar al restaurante sobre un nuevo pedido recibido.
- RF16: El sistema debe permitir al restaurante aceptar, rechazar o actualizar el estado del pedido.
- RF17: El sistema debe permitir a los repartidores recibir pedidos disponibles y aceptar la entrega.
- RF18: El sistema debe calcular el precio total de un pedido, incluyendo impuestos, tarifas de servicio y costos de entrega.
- RF19: El sistema debe aplicar descuentos o cupones promocionales cuando correspondan.
- RF20: El sistema debe permitir pagos en línea mediante tarjeta de crédito, débito, billeteras digitales y efectivo contra entrega.
- RF21: El sistema debe validar y confirmar las transacciones realizadas por los clientes.
- RF22: El sistema debe notificar al cliente y al restaurante cuando el pago sea exitoso o rechazado.

2,3)




4 ) Justificación,

He decidido dividir el sistema en piezas clave que se encargan de tareas específicas. Por ejemplo, tengo un módulo solo para el registro y autenticación de usuarios, otro para la búsqueda de productos y otro para gestionar el carrito de compras. Cada una de estas piezas funciona de manera independiente, lo que me facilita que el sistema crezca y se adapte sin problemas.

Además, he definido bien la autenticación y los roles, lo que significa que los clientes solo pueden acceder a las funcionalidades que necesitan para pedir comida, mientras que los restaurantes y repartidores tienen acceso a las funcionalidades que les permiten gestionar pedidos y entregas. Me gusta que el carrito de compras y el procesamiento de pedidos estén separados, porque esto me permite reutilizar el carrito en otros contextos, como compras en supermercados. También he decidido manejar los cálculos de precios y procesamiento de pagos en servicios específicos, lo que garantiza que las tarifas, impuestos y descuentos se calculen correctamente y que las transacciones sean seguras. Todo esto lo he diseñado para que sea escalable y extensible, lo que significa que puedo agregar nuevos servicios y funcionalidades en el futuro, como integrar pasarelas de pago adicionales o servicios de geolocalización para rastrear los pedidos.

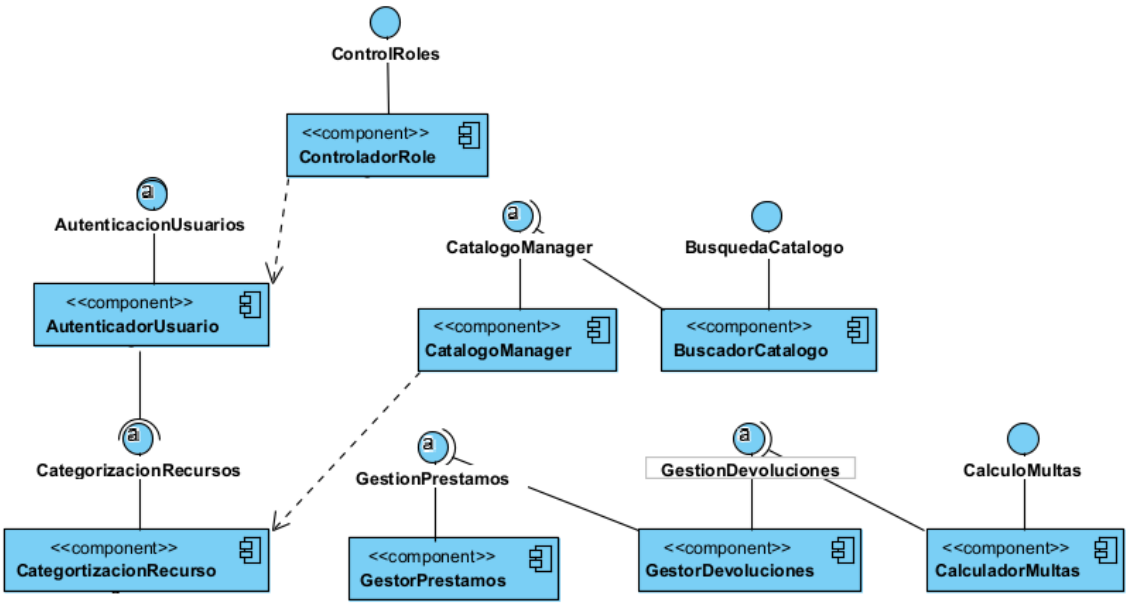
## 3) Requerimientos Funcionales

- RF1: El sistema debe permitir registrar, actualizar y eliminar libros y materiales de la biblioteca.
- RF2: El sistema debe permitir la consulta general del catálogo por parte de los usuarios autorizados.
- RF3: El sistema debe permitir a los usuarios buscar libros por título, autor, ISBN, editorial o palabras clave.
- RF4: El sistema debe mostrar disponibilidad del recurso (ejemplar disponible o prestado).
- RF5: El sistema debe permitir clasificar los recursos por categorías (ejemplo: literatura, ciencia, historia, matemáticas).
- RF6: El sistema debe permitir asociar etiquetas temáticas a los recursos para mejorar la búsqueda.
- RF7: El sistema debe permitir a los usuarios autorizados (estudiantes/profesores) solicitar el préstamo de un recurso disponible.
- RF8: El sistema debe registrar fecha de inicio y fecha de devolución del préstamo.
- RF9: El sistema debe limitar el número de préstamos activos por usuario según las políticas de la institución.

- RF10: El sistema debe permitir a los usuarios registrar la devolución de un recurso prestado.
- RF11: El sistema debe actualizar automáticamente el estado del recurso en el catálogo al ser devuelto.
- RF12: El sistema debe permitir registrar devoluciones tardías para generar cálculos de multas.
- RF13: El sistema debe calcular automáticamente las multas por devoluciones atrasadas en función de los días de retraso.
- RF14: El sistema debe notificar al usuario del monto de la multa pendiente al devolver el recurso.
- RF15: El sistema debe generar reportes de multas pendientes por usuario.
- RF16: El sistema debe permitir que los usuarios se autentiquen mediante credenciales seguras.
- RF17: El sistema debe validar si un usuario está registrado en el colegio antes de conceder acceso.

- RF18: El sistema debe gestionar diferentes roles (administrador, profesor, estudiante, bibliotecario).
- RF19: El sistema debe otorgar permisos diferenciados según el rol (ejemplo: solo administradores pueden eliminar recursos, los estudiantes solo pueden solicitar préstamos).

2,3)




4) Justificación.

Antes, mi sistema de gestión de biblioteca era un monolito que mezclaba todo en un solo módulo, lo que hacía que fuera muy difícil de mantener. Pero ahora he decidido dividir las responsabilidades en componentes especializados, como la gestión de catálogo, préstamos, devoluciones, multas, autenticación y roles. Esto sigue el principio de alta cohesión y bajo acoplamiento, lo que significa que cada componente se enfoca en una tarea específica y no depende demasiado de los demás.

En cuanto a la gestión de usuarios y seguridad, he creado componentes separados para la autenticación y el control de roles. Esto me permite diferenciar la validación de credenciales de la asignación de permisos, lo que facilita la ampliación futura, como integrar login con Google o asignar permisos especiales para bibliotecarios.

Mi catálogo ahora está modularizado en tres componentes: CatalogoManager, BuscadorCatalogo y CategorizacionRecursos. El CatalogoManager administra los recursos, el BuscadorCatalogo optimiza las consultas y la CategorizacionRecursos añade flexibilidad para clasificar y etiquetar materiales. Esto mejora la escalabilidad y me permite tener un control preciso sobre el catálogo.

El flujo de préstamos y devoluciones también ha sido optimizado gracias a los componentes GestorPrestamos y GestorDevoluciones. Estos componentes permiten llevar un control preciso del ciclo de vida de los recursos y se conectan directamente con el CatalogoManager para actualizar la disponibilidad en tiempo real.

Además, he desacoplado el cálculo de multas en un componente independiente, lo que me permite cambiar las reglas de negocio sin afectar otros módulos. Esto es especialmente útil si quiero implementar diferentes políticas de multa según el rol del usuario o por tipo de recurso.

La modularización también facilita la integración con otros sistemas, como reportes estadísticos, notificaciones por correo o integración con pagos en línea. Cada componente puede evolucionar de forma independiente, lo que reduce la deuda técnica que tenía el sistema monolítico y me permite tener un sistema más escalable y mantenible.