**Yeraldin Araujo Urbano**

**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.

* RF-01 El sistema debe permitir registrar nuevos pacientes con información personal, historial médico y datos de contacto.
* RF-02 El sistema debe permitir consultar, editar y eliminar la información de los pacientes registrados.
* RF-03 El sistema debe permitir a los pacientes reservar citas médicas según disponibilidad de los profesionales de salud.
* RF-04 El sistema debe permitir a los médicos configurar y administrar su agenda de consultas, incluyendo horarios disponibles y bloqueos.
* RF-05 El sistema debe permitir la creación, edición y eliminación de usuarios con distintos roles (administrador, médico, paciente).
* RF-06 El sistema debe implementar autenticación y autorización para garantizar el acceso seguro según el rol del usuario.

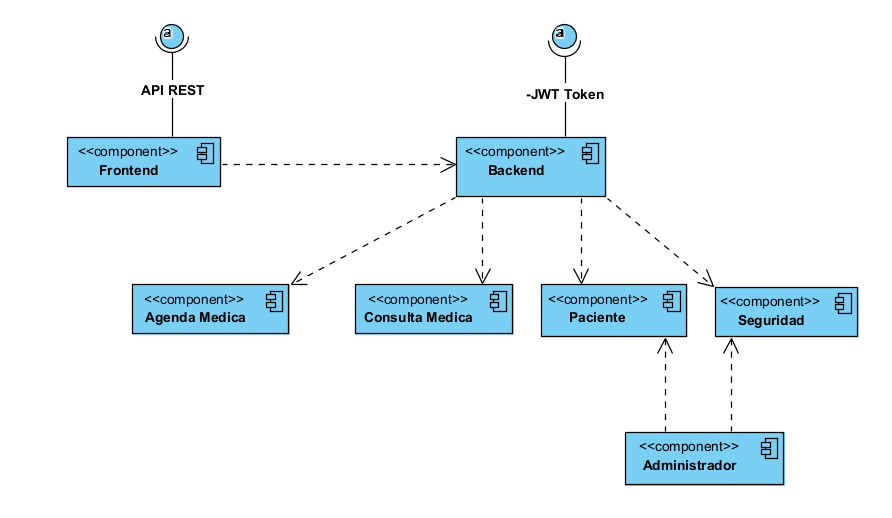
|  |  |
| --- | --- |
| Funcionalidad | Componentes |
| Registro y gestión de pacientes | Pacientes |
| Reserva de consultas médicas | Consulta Medica |
| Gestión de agenda por parte de los médicos | Agenda Medica |
| Administración de usuarios y control del sistema | Administrador |

Interfaces – Conectores

* API REST para comunicación entre frontend y backend
* Base de datos relacional (MySQL/PostgreSQL)
* Autenticación mediante tokens (JWT)
* Interfaz web responsiva para pacientes, médicos y administradores

Argumentos de diseño

* Separar la agenda médica de las consultas permite flexibilidad en la programación.
* El módulo de administración centraliza el control sin interferir en la experiencia del usuario.
* Uso de APIs facilita escalabilidad y futuras integraciones (por ejemplo, con sistemas de historia clínica).



**Ejercicio 2.** Un 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.

* RF-01 El sistema debe permitir registrar usuarios con roles diferenciados: cliente, restaurante y repartidor.
* RF-02 El sistema debe permitir a los restaurantes crear perfiles con información de contacto, ubicación y menú disponible.
* RF-03 El sistema debe permitir a los repartidores registrarse y vincularse a zonas de entrega específicas.
* RF-04 El sistema debe permitir a los clientes consultar el menú de cada restaurante con precios, imágenes y disponibilidad.
* RF-05 El sistema debe permitir a los clientes seleccionar productos del menú y generar pedidos con opciones de pago.
* RF-06 El sistema debe permitir a los repartidores recibir asignaciones de pedidos y actualizar el estado de la entrega en tiempo real.
* RF-07 El sistema debe permitir a los clientes visualizar el estado de su pedido desde la confirmación hasta la entrega.
* RF-08 El sistema debe permitir realizar pagos electrónicos seguros y registrar la transacción asociada al pedido.

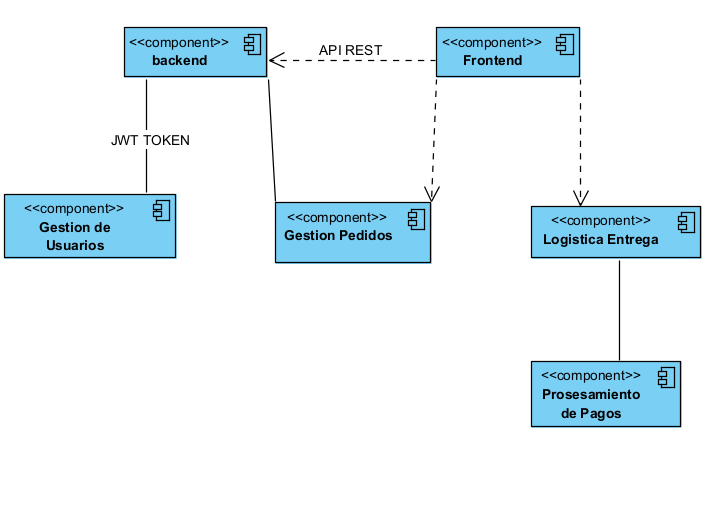
|  |  |
| --- | --- |
| Funcionalidad | Componentes |
| Registro de usuarios, restaurantes y repartidores | Gestión de usuarios |
| Visualización de menú y pedidos | Gestión de pedidos |
| Gestión de entregas en tiempo real | Logística de entregas |
| Pagos y seguimiento de pedidos | Procesamiento |

Interfaces – Conectores

* API REST + WebSockets para actualizaciones en tiempo real
* Integración con pasarelas de pago (ej. PayU, Stripe)
* Mapas interactivos (Google Maps API)
* Aplicaciones móviles (Android/iOS) y web

Argumentos de diseño

* Separar usuarios, restaurantes y repartidores permite personalizar la experiencia según el rol.
* WebSockets mejoran la experiencia en tiempo real (seguimiento de pedidos).
* Modularizar pagos y logística facilita la integración con servicios externos.



**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.

* RF-01 El sistema debe permitir registrar, consultar, renovar y devolver préstamos de libros, vinculando cada operación al usuario correspondiente.
* RF-02 El sistema debe permitir crear, editar y eliminar perfiles de usuarios (estudiantes, docentes, administradores), incluyendo control de acceso por roles.
* RF-03 El sistema debe permitir consultar, agregar, editar y eliminar registros de libros, incluyendo información como título, autor, género, disponibilidad y ubicación física.
* RF-04 El sistema debe permitir generar reportes sobre préstamos activos, historial de préstamos, libros más solicitados y estadísticas de uso por usuario o categoría.

|  |  |
| --- | --- |
| Funcionalidad | Componentes |
| Gestión de préstamos | Prestamos |
| Administración de usuarios | Usuarios |
| Catálogo de libros | Catalogo |
| Generación de reportes | Reportes |

Interfaces – Conectores

* API RESTful para desacoplar lógica y presentación
* Base de datos normalizada
* Interfaz web para usuarios y administradores
* Exportación de reportes en PDF/Excel

Argumentos de diseño

* Separar el catálogo de los préstamos mejora la mantenibilidad.
* Modularizar reportes permite generar estadísticas sin afectar el rendimiento del sistema.
* Refactorizar con principios SOLID y MVC facilita futuras actualizacióne