Ejercicio 1

1. Requerimientos funcionales

ID	Nombre	Descripción	Actor	Prioridad
RF-01	Registro de pacientes	El sistema debe permitir registrar un paciente con datos personales y de contacto.	Paciente	Alta
RF-02	Reserva de cita	El sistema debe permitir que un paciente reserve una cita médica seleccionando médico, fecha y hora.	Paciente	Alta
RF-03	Gestión de agenda	El sistema debe permitir a los médicos gestionar su agenda, incluyendo creación, modificación y cancelación de horarios.	Médico	Alta
RF-04	Notificaciones	El sistema debe enviar notificaciones de confirmación, recordatorio o cancelación de cita al paciente y al médico.	Sistema	Media
RF-05	Administración de usuarios	El sistema debe permitir a los administradores gestionar usuarios y roles.	Administrador	Alta

2. Modularización del sistema

Módulos identificados:

1. Módulo de Autenticación y Roles → registro, login, gestión de roles.
2. Módulo de Pacientes → funcionalidades específicas para usuarios pacientes.
3. Módulo de Médicos y Agenda → gestión de horarios y disponibilidad.
4. Módulo de Citas y Reservas → reserva, cancelación y confirmación.
5. Módulo de Administración → gestión de usuarios, roles y reportes.
6. Módulo de Notificaciones → envío de recordatorios vía correo/SMS.

3. Interfaces – Conectores

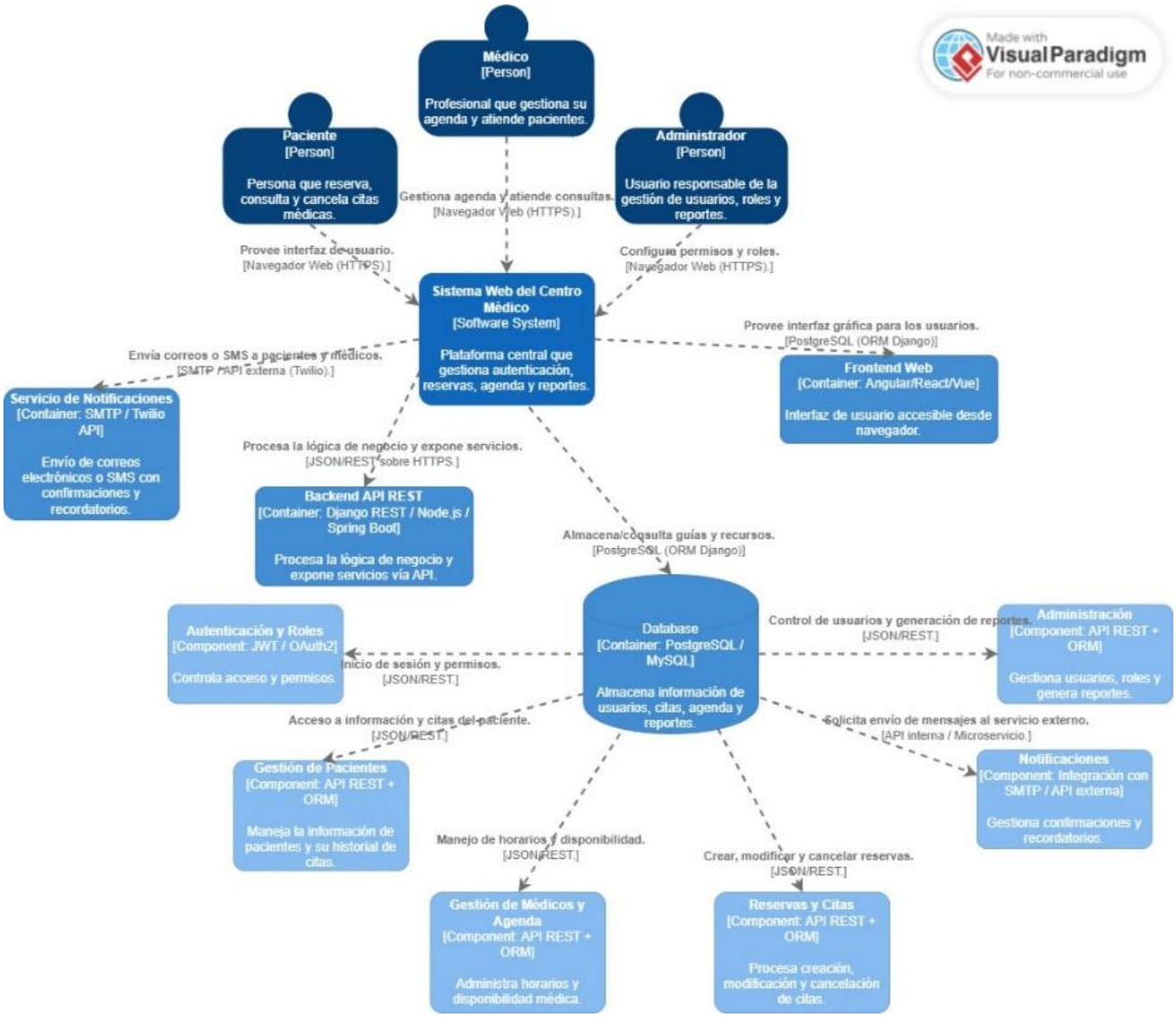
- **Interfaces internas (Backend ↔ BD):**
    - Consultas SQL para almacenar usuarios, horarios, citas y reportes.
- **Interfaces externas (Frontend ↔ Backend):**
    - **API RESTful con endpoints como:**

        - /api/auth/login
        - /api/patients/{id}/appointments
        - /api/doctors/{id}/agenda
        - /api/admin/reports
- **Conectores externos:**
    - Servicio de correo electrónico (SMTP) o SMS para notificaciones.

4. Justificación del diseño

La modularización propuesta sigue principios de **separación de responsabilidades**: cada módulo cumple una función concreta (p. ej., autenticación aislada de agenda médica). Esto garantiza:

- **Escalabilidad:** si el sistema crece, se pueden añadir más especialidades o nuevos roles sin afectar la lógica principal.
- **Mantenibilidad:** el módulo de notificaciones puede sustituirse (ej: cambiar de SMS a WhatsApp API) sin alterar la gestión de citas.
- **Seguridad:** el módulo de autenticación centraliza el control de accesos y privilegios.
- **Eficiencia:** separar reportes y agenda reduce la complejidad de consultas y mejora el rendimiento



Ejercicio 2

1. Requerimientos funcionales

ID	Nombre	Descripción	Actor	Prioridad
RF-01	Registro de usuarios	El sistema debe permitir registrar clientes, restaurantes y repartidores.	Cliente / Restaurante / Repartidor	Alta
RF-02	Autenticación y roles	El sistema debe permitir iniciar sesión y asignar permisos según el rol.	Todos	Alta
RF-03	Gestión de menús	El sistema debe permitir a los restaurantes crear, modificar y eliminar menús y productos.	Restaurante	Alta

RF-04	Pedido de comida	El sistema debe permitir a los clientes realizar pedidos seleccionando restaurante, menú y método de pago.	Cliente	Alta
RF-05	Logística de entregas	El sistema debe permitir asignar pedidos a repartidores y realizar seguimiento de la entrega.	Repartidor / Sistema	Alta
RF-06	Procesamiento de pagos	El sistema debe permitir pagos en línea de manera segura.	Cliente / Sistema	Alta
RF-07	Notificaciones en tiempo real	El sistema debe enviar notificaciones del estado del pedido a clientes, restaurantes y repartidores.	Sistema	Media
RF-08	Administración de usuarios	El sistema debe permitir al administrador gestionar cuentas y generar reportes.	Administrador	Alta

2. Modularización del sistema

Módulos identificados:

1. **Módulo de Autenticación y Roles** → Registro, login y permisos.
2. **Módulo de Gestión de Usuarios y Perfiles** → Manejo de clientes, restaurantes y repartidores.
3. **Módulo de Restaurantes y Menús** → Administración de menús y disponibilidad.
4. **Módulo de Pedidos** → Creación, actualización y cancelación de pedidos.
5. **Módulo de Logística de Repartidores** → Asignación y seguimiento de entregas.
6. **Módulo de Pagos** → Procesamiento de transacciones en línea.
7. **Módulo de Notificaciones** → Envío de alertas en tiempo real.
8. **Módulo de Administración** → Control de usuarios, cuentas y reportes.

3. Interfaces – Conectores

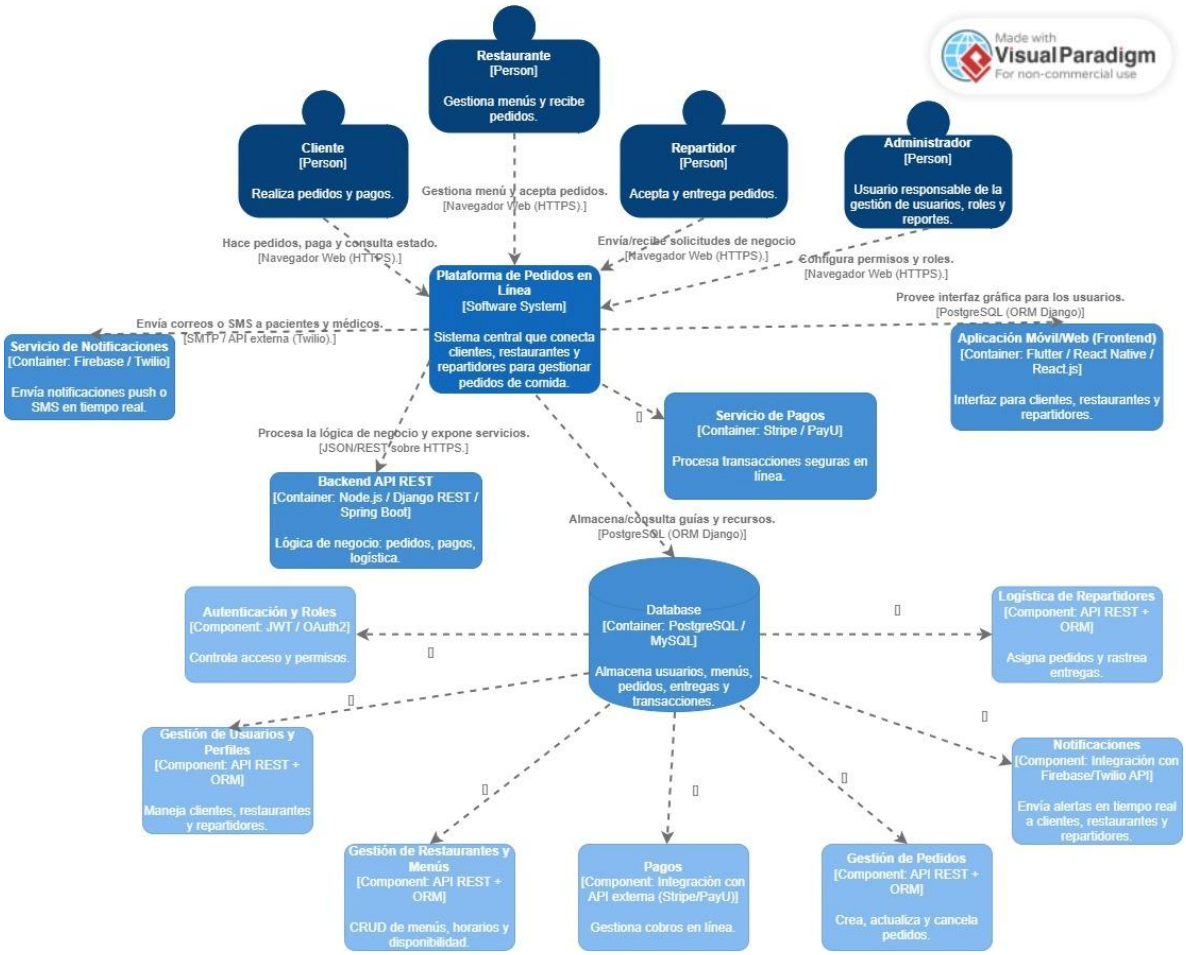
- **Interfaces internas (Backend ↔ BD):**
  - Consultas SQL para almacenar usuarios, menús, pedidos, repartidores y transacciones.
- **Interfaces externas (Frontend ↔ Backend):**
  - API RESTful con endpoints como:
    - /api/auth/login
    - /api/customers/{id}/orders
    - /api/restaurants/{id}/menu
    - /api/drivers/{id}/deliveries
    - /api/admin/reports

- **Conectores externos:**
  - Servicio de pagos (Stripe, PayU).
  - Servicio de notificaciones push o SMS (Firebase, Twilio).
  - Servicio de geolocalización (Google Maps API).

4. Justificación del diseño

La modularización propuesta se fundamenta en la **separación de responsabilidades** y la necesidad de escalar el sistema:

- **Escalabilidad:** cada módulo puede evolucionar de manera independiente (ejemplo: pagos podría migrar de PayU a Stripe sin afectar pedidos).
- **Mantenibilidad:** separar restaurantes, pedidos y logística reduce la complejidad y facilita la depuración.
- **Disponibilidad:** el sistema puede crecer de forma distribuida, permitiendo balancear carga en módulos críticos como pedidos y notificaciones.
- **Seguridad:** el módulo de autenticación garantiza que cada usuario solo acceda a funcionalidades propias de su rol.
- **Experiencia de usuario:** el módulo de notificaciones asegura comunicación fluida y en tiempo real con clientes, restaurantes y repartidores.



Ejercicio 3

1. Requerimientos funcionales

ID	Nombre	Descripción	Actor	Prioridad
RF-01	Registro de usuarios	El sistema debe permitir registrar estudiantes, docentes y bibliotecarios.	Bibliotecario	Alta
RF-02	Autenticación y roles	El sistema debe permitir iniciar sesión y validar permisos por rol.	Todos	Alta
RF-03	Gestión de catálogo	El sistema debe permitir administrar libros, autores y categorías.	Bibliotecario	Alta
RF-04	Consulta de catálogo	El sistema debe permitir a los usuarios buscar y visualizar libros disponibles.	Estudiante / Docente	Alta
RF-05	Préstamos de libros	El sistema debe permitir registrar préstamos de libros a usuarios.	Bibliotecario	Alta
RF-06	Devoluciones de libros	El sistema debe permitir registrar devoluciones de libros.	Bibliotecario	Alta
RF-07	Control de sanciones	El sistema debe calcular y aplicar sanciones a usuarios que no devuelvan los libros en el tiempo establecido.	Sistema	Media
RF-08	Generación de reportes	El sistema debe generar reportes de préstamos, devoluciones y sanciones.	Bibliotecario / Administrador	Media
RF-09	Notificaciones	El sistema debe enviar alertas de vencimientos y sanciones a los usuarios.	Sistema	Media

2. Modularización del sistema

Módulos identificados:

1. **Módulo de Autenticación y Roles** → Login y control de accesos.
2. **Módulo de Usuarios** → Registro y gestión de estudiantes, docentes y bibliotecarios.
3. **Módulo de Catálogo** → CRUD de libros, categorías y autores.
4. **Módulo de Préstamos y Devoluciones** → Registro de operaciones de préstamo y devolución.
5. **Módulo de Sanciones** → Cálculo y aplicación de penalizaciones.
6. **Módulo de Reportes** → Generación de estadísticas y reportes en PDF/Excel.
7. **Módulo de Notificaciones (opcional)** → Envío de alertas por correo o push.

3. Interfaces – Conectores

- **Interfaces internas (Backend ↔ BD):**
  - Consultas SQL para usuarios, libros, préstamos, devoluciones y sanciones.
- **Interfaces externas (Frontend ↔ Backend):**
  - API RESTful con endpoints como:
    - /api/auth/login
    - /api/catalog/books
    - /api/loans/create
    - /api/returns/{id}
    - /api/reports
- **Conectores externos:**
  - Servicio de correo electrónico (SMTP) o notificaciones push.

4. Justificación del diseño

El rediseño del sistema aplica principios de **modularización** para reemplazar el enfoque monolítico:

- **Escalabilidad:** el catálogo puede ampliarse para incluir libros digitales sin modificar préstamos o devoluciones.
- **Mantenibilidad:** separar sanciones y reportes reduce la complejidad del código y facilita cambios futuros.
- **Eficiencia:** los reportes se alimentan de distintos módulos, pero su independencia mejora el rendimiento.
- **Seguridad:** centralizar la autenticación y roles garantiza un control de acceso más confiable.
- **Flexibilidad:** el módulo de notificaciones es opcional, lo que permite integrar servicios adicionales (ej. WhatsApp API) sin afectar la lógica principal.