UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: AGOSTO 2025 – ENERO 2026

Prueba práctica

I. PORTADA

Tema:	Preguntas de la prueba
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Nivel - Paralelo
Alumnos participantes:	JONATHAN ISRAEL LOZADA LOPEZ ……
Asignatura:	Aplicaciones Distribuidas
Docente:	Ing. Rubén Caiza, Mg.

Grupo: Manzano, Ortiz, Rodríguez, Ayuquina

1. ¿Qué tipo de arquitectura utiliza el sistema?
Respuesta: Arquitectura de microservicios con 6 servicios especializados y 3 bases de datos MySQL distribuidas por región (Central, Guayaquil, Cuenca).
2. ¿Cuáles son las dos interfaces principales del frontend?
Respuesta: Interfaz Administrativa (admin) e Interfaz de Hospital (médico), con navegación diferenciada por roles y accesos restringidos.
3. ¿Cómo se gestiona la seguridad en el sistema?
Respuesta: Autenticación JWT distribuida, protección por roles, validación automática de tokens, y permisos específicos por centro médico.
4. ¿Cuántos puertos diferentes usa el sistema?
Respuesta: 8 puertos: Frontend (5173), API Gateway (3000), Auth Service (3001), Admin Service (3002), Consultas Service (3003), Users Service (3004), Reports Service (3005), y las bases de datos en puertos 3307-3309.
5. ¿Cómo se despliega y ejecuta el proyecto?
Respuesta: Docker Compose automatizado con 9 contenedores, cada servicio con su puerto específico y bases de datos distribuidas, comunicación interna segura.

Grupo: Barragán, Bonilla, García, Manjarres

1. ¿Qué tipo de arquitectura utiliza el sistema?

Respuesta:
El sistema utiliza una arquitectura de microservicios, conformada por servicios independientes como autenticación, administración, gestión médica y reportes, coordinados mediante Spring Cloud Gateway y Eureka Server para el enrutamiento y descubrimiento de servicios.

2. ¿Cómo se garantiza la seguridad dentro del sistema?

Respuesta:
La seguridad se implementa mediante autenticación JWT gestionada por el microservicio de autenticación. El API Gateway valida los tokens antes de enrutar las peticiones, y cada microservicio aplica control de acceso por roles (ADMIN y DOCTOR), asegurando la protección de datos y operaciones sensibles.

3. ¿De qué manera se comunican los microservicios entre sí?

Respuesta:
Los microservicios se comunican de forma síncrona mediante APIs RESTful, registrándose dinámicamente en Eureka Server. El API Gateway centraliza las solicitudes externas y las enruta al servicio correspondiente, propagando cabeceras internas para mantener la identidad y los permisos del usuario.

4. ¿Cómo se gestiona el despliegue y la ejecución del proyecto?

Respuesta:
El sistema se despliega mediante Docker Compose, que orquesta todos los contenedores del ecosistema: base de datos PostgreSQL, los microservicios del backend, el servidor Eureka, el API Gateway y el frontend React con Nginx. Con un solo comando (docker compose up) se levanta todo el entorno de forma automatizada y consistente.

5. ¿Cuáles son las principales interfaces del frontend y qué roles manejan?

Respuesta:
El frontend, desarrollado en React con Vite y TailwindCSS, ofrece dos interfaces principales:

Interfaz Administrativa, destinada a los usuarios con rol ADMIN, para gestionar doctores, centros médicos, especialidades y reportes.

Interfaz Médica, diseñada para usuarios con rol DOCTOR, enfocada en la gestión de pacientes, consultas médicas y especialidades del centro asignado.

Grupo: Cuadrado, Lozada, Rubio, Serrano

1) ¿Qué tipo de arquitectura usa el proyecto?
Respuesta: Microservicios separados por dominios (Consultas y Administración) comunicados principalmente por gRPC.

2) ¿Cómo se autentican las solicitudes entre servicios?
Respuesta: Con JWT Bearer; los métodos llevan [Authorize] y se validan Issuer/Audience/Signature.

3) ¿Qué rol cumple el microservicio de Administración?
Respuesta: Centraliza catálogos y seguridad (Usuarios, Empleados, Especialidades, Centros, Tipos de empleado).

4) ¿Cómo se persisten los datos?
Respuesta: Con EF Core y MySQL; cada microservicio tiene su DbContext y migraciones propias.

5) ¿Cómo consume un frontend el microservicio de Consultas si el navegador no soporta gRPC nativo?
Respuesta: Usando gRPC-Web habilitado en el backend o un “puente” REST que invoque internamente gRPC.

Grupo: Guachi, Luisa, Villavicencio

1) ¿Qué arquitectura de microservicios implementaste y cómo garantizaste el desacoplamiento entre ellos?
Se implementaron microservicios independientes para administración, consultas médicas y gestión de usuarios, comunicados mediante APIs REST. Cada microservicio tiene su propia base de datos o acceso específico, lo que permite escalar de forma independiente.

2) ¿Cómo aseguraste la comunicación entre microservicios?
Se implementó un API Gateway que centraliza las peticiones desde el frontend hacia los diferentes microservicios.

3) ¿Cómo resolviste el despliegue de las aplicaciones en la nube?
Se desplegaron las aplicaciones en Azure App Services para el frontend y backend, y las bases de datos en Azure Virtual Machines con MariaDB.

4) ¿Qué herramientas utilizaste para la documentación técnica del proyecto?
Se utilizó Swagger para documentar los endpoints de las APIs, diagramas de arquitectura para representar los microservicios y su interacción, y Azure DevOps para almacenar y compartir toda la documentación del proyecto.

5) ¿Qué diferencia hubo entre la interfaz de administración y la interfaz de hospitales?
La interfaz de administración gestionar médicos, empleados, especialidades y centros médicos (mediante el consumo de las apis).
La interfaz de hospitales permite a los médicos crear, consultar y modificar consultas médicas, con autenticación de usuario para limitar el acceso a su propio hospital.

Grupo: Chavez , Giler , Hurtado, Velastegui

1 ¿Por qué se utilizo un API Gateway en una arquitectura de microservicios?
El API Gateway actúa como un punto de entrada único para todas las solicitudes del cliente.
2 ¿Qué ventajas tiene usar gRPC entre microservicios en lugar de REST?
gRPC ofrece comunicación binaria y basada en contratos (.proto), lo que lo hace más eficiente que REST en JSON.
3 ¿Por qué el proyecto utiliza replicación de bases de datos entre la central y las clínicas locales?
La replicación permite sincronizar automáticamente los datos globales (como los médicos o empleados) desde la base de datos central hacia las bases locales.
4 ¿Cuál es la responsabilidad principal del microservicio de autenticación?
El microservicio de autenticación se encarga de gestionar el acceso de los usuarios.
5 ¿Cómo se asegura la comunicación entre los microservicios en este proyecto?
La comunicación entre los microservicios se asegura mediante el uso de tokens JWT (JSON Web Tokens) generados por el Microservicio de Autenticación.

Grupo: Barros, Jijon, Lopez, Paredes

1) ¿Qué arquitectura se usó para cada microservicio?

R: Se uso una arquitectura hexagonal para cada microservicio, donde se diferencias las capas de acceso a datos, dominio, presentación e infraestructura.

2) ¿Qué arquitectura de microservicios se usó para desacoplar y comunicar los servicios entre sí?

R: Se uso una arquitectura de API Gateway, donde se centraliza el acceso a los microservicios por parte del frontend y ofrece un punto único de acceso.

3) ¿Mediante qué estándar se maneja la autenticación y autorización del acceso a los recursos?

R: La autenticación y autorización se maneja mediante el estándar RFC 7519, donde se envían tokens de información como objetos JSON en la cabecera de los pedidos. Este estándar también se llama JWT (JSON Web Tokens).

4) ¿Qué tecnologías se usaron para el proyecto?

R: Se utilizó React para el frontend, NodeJS para el API Gateway, NodeJS con Typescript y Javascript para los microservicios y Prisma con MySQL para la base de datos.

5) ¿Cómo se despliega el proyecto?

R: El proyecto se despliega en máquinas virtuales independientes, donde el acceso a los servicios ofrecidos por cada máquina se puede configurar mediante las variables de entorno del servicio.

Grupo: Constante, Conterón, García, Montero

1. Qué relación existe entre usuarios del sistema con doctores?
Todo doctor es un usuario del sistema al tener credenciales de acceso, a excepción del admin que no es doctor pero si un usuario con diferente rol.

2. Cuántas interfaces tiene el sistema?
Se tiene dos interfaces: una de admin con todas las operaciones Crud disponibles y la de médicos para gestionar sus citas.

3. Cuántos micro servicios ofrece el backend?
Tres micro servicios: servicio de administración, de médicos y de autenticación.

4. Qué herramienta de documentación se usa para exponer los endpoints?
Se hace uso de Scalar herramienta alternativa a Swagger más moderna.

5. ¿Qué beneficios aporta la contenerización con Docker en este proyecto?
El sistema está completamente contenerizado con Docker, asignando un contenedor para cada microservicio del backend, otro para el API Gateway y uno para cada interfaz del frontend (Admin-UI y Hospital-UI). Esta estructura facilita el despliegue, asegura el aislamiento de servicios y permite una arquitectura distribuida más escalable y mantenible.

Grupo: Chimborazo, Punina, Sánchez,Tunja

1. ¿ Cuál es el objetivo principal del proyecto desarrollado en la prueba práctica?
Respuesta: El objetivo general es implementar una arquitectura distribuida para la gestión de hospitales y consultas médicas, aplicando conceptos de microservicios, bases de datos distribuidas y comunicación entre sistemas.

2.¿Qué componentes principales conforman la arquitectura distribuida implementada?
Respuesta: La arquitectura está compuesta por un API Gateway central que controla la comunicación entre los diferentes módulos, microservicios por sede (Quito, Guayaquil, Cuenca) con su propio backend y base de datos MongoDB local, y un frontend desarrollado en Next.js y Vue.js para la interfaz hospitalaria de cada ciudad.

3. ¿Qué tecnologías y herramientas se utilizaron en el desarrollo del sistema?
Respuesta: Se utilizaron Node.js (microservicios con NestJS) para el backend, MongoDB con réplica distribuida mediante Docker Compose para la base de datos, Next.js y Vue.js para el frontend, JWT para la autenticación, Swagger / OpenAPI para la documentación y package.json para la gestión de dependencias.

4. ¿Qué beneficios ofrece la arquitectura distribuida en este sistema hospitalario?
Respuesta: Permite descentralizar la gestión de datos, mejorar la disponibilidad y la escalabilidad del sistema, facilita el mantenimiento al separar los microservicios y centraliza la seguridad y el control de accesos a través del API Gateway.

5. ¿Cómo se realizó el despliegue de la base de datos distribuida?

Respuesta: El despliegue se realizó mediante un archivo Docker Compose que define tres servicios de MongoDB (un nodo primario y dos secundarios) configurados en un conjunto de réplicas (replica set). Esta configuración garantiza alta disponibilidad, tolerancia a fallos y comunicación entre los nodos dentro de una red personalizada llamada mongo-cluster.
vele si te parecen estas preguntas