Actividad 6 - Aplicando Arquitecturas de Software

para Soluciones Empresariales Modernas

Jhon Andersson Vanegas Castañeda

Jorge Castañeda

Arquitectura de Software

Universidad Iberoamericana

Bogotá
05/10/2025

Contenido

Análisis de Requisitos y Diseño de la Arquitectura

1. **Identificar Objetivos de la Aplicación:**

El desarrollo propuesto tiene como objetivo el automatizar y optimizar procesos de los académicos, administrativos y soportes de estudiantiles dentro de la Corporación Universitaria de Asturias, garantizando un proceso eficiente, escalable y de manera segura. A continuación, detallaremos los requisitos funcionales y no funcionales que sean claves, enfocados en los principios de escalabilidad, resiliencia y seguridad.

2. **Requisitos Funcionales**

Gestión de Usuarios y Roles

- Permitir el registro, autenticación y administración de usuarios (estudiantes, docentes, administrativos y directivos).

Gestión Académica

- Automatizar los procesos de matrícula, actualización de datos, generación de actas, reportes y certificados.
- Controlar los expedientes académicos de cada estudiante, incluyendo notas, asignaturas y seguimiento de progreso.

Gestión de Solicitudes (PQRS y trámites)

- Permitir el registro y seguimiento de solicitudes mediante un formulario por parte de los estudiantes, egresados, proveedores y aspirantes.
- Automatizar la asignación, trazabilidad y cierre de cada caso mediante flujos de trabajo según el tipo de solicitante y tipo de solicitud o tramite.

Gestión de Mentorías y Seguimiento

- Asignar mentores automáticamente a cada estudiante con el fin de generar un seguimiento continuo en los procesos de este.

- Registrar sesiones de acompañamiento académico y generar reportes de seguimiento.

Notificaciones y Comunicaciones

- Enviar correos electrónicos y alertas automáticas tanto como estudiantes, solicitantes, mentores, ante eventos importantes (resolución de PQRS, aprobación de trámites, vencimientos, seguimientos y procesos académicos).
- Integrar plantillas personalizadas mediante servicios de mensajería (Notifications Service).

Reportes e Indicadores

- Generar paneles de control (dashboards) con métricas de los procesos académicos académicos y administrativas.
- Reportes de las solicitudes PQRS y resoluciones de estas.
- Exportar reportes automáticos en formato PDF o Excel para análisis institucional

3. Requisitos no Funcionales:

Desempeño:

El sistema tiene que responder a las solicitudes de los usuarios en menos de 3 segundos para tareas comunes, como consultar notas, realizar matrículas o hacer pagos.

Disponibilidad:

El sistema debe garantizar un tiempo de disponibilidad incluyendo el mantenimiento programado fuera de las horas pico.

Seguridad

- Autenticación y autorización basadas en roles (RBAC).
- Cifrado de contraseñas con BCrypt y manejo seguro de tokens JWT.
- Protección de endpoints con HTTPS, control de CORS y validación de cabeceras.
- Políticas de backup y recuperación ante desastres, junto con auditoría de eventos del sistema.

- Cumplimiento con normativas de protección de datos (Ley 1581 de 2012 – Habeas Data en Colombia).

Escalabilidad:

El sistema debe ser capaz de adaptarse al crecimiento de estudiantes, docentes y programas sin que se degrade el rendimiento. También debe permitir la adición de módulos o funcionalidades, como nuevos programas académicos, módulos de mentoría o integración con herramientas externas, sin afectar la operación existente.

Mantenibilidad:

El sistema debe ser modular y estar bien documentado, lo que permitirá realizar actualizaciones y mejoras con un impacto mínimo en las operaciones.

Resiliencia

- Implementación de **circuit breakers (Resilience4j)** y políticas de reintento con backoff exponencial.
- Configuración de **health checks** y **auto recuperación** (self-healing) para los servicios desplegados.
- Replicación de base de datos y backups automáticos para garantizar continuidad operativa.

Interoperabilidad

- Comunicación estándar entre servicios mediante **REST API** y formato **JSON**.
- Registro de microservicios con Eureka Server
- Autenticación distribuida mediante **JWT (JSON Web Tokens)** compartido entre microservicios.
- Posibilidad de integración futura con sistemas externos (por ejemplo, CRM, SRM y herramientas como Teams) mediante API Gateway.

Escalabilidad

- La arquitectura de microservicios permite escalar cada componente de manera independiente según se requiera desarrollar.
- Uso de **Kubernetes con Horizontal Pod Autoscaler (HPA)** para aumentar instancias automáticamente.
- Integración con balanceadores de carga (Spring Cloud Gateway) para distribuir el tráfico y recursos de los microservicios.

4. Diagrama de Arquitectura de Software:

Microservicios:

- **Auth Service (JWT-AUTH)**
  - Emisión y refresh de JWT; validación, gestión de sesiones.
  - Interactúa con Users DB para credenciales.
- **Users Service**
  - CRUD de usuarios, roles y permisos.
  - Base de datos perteneciente al microservicio en POSTRESQL (Users DB).
- **Request Service (Gestión de solicitudes / PQRS / trámites y procesos Académicos.**
  - Lógica de negocio de trámites, flujos, estados, reglas.
  - Manejo de archivos.
  - Base de datos propia (Request DB).

- **Notifications Service**
    - Plantillas, envío de emails (SMTP con Outlook), colas para envío asíncrono
    - Envió de correos a procesos generados desde otro microservicio.

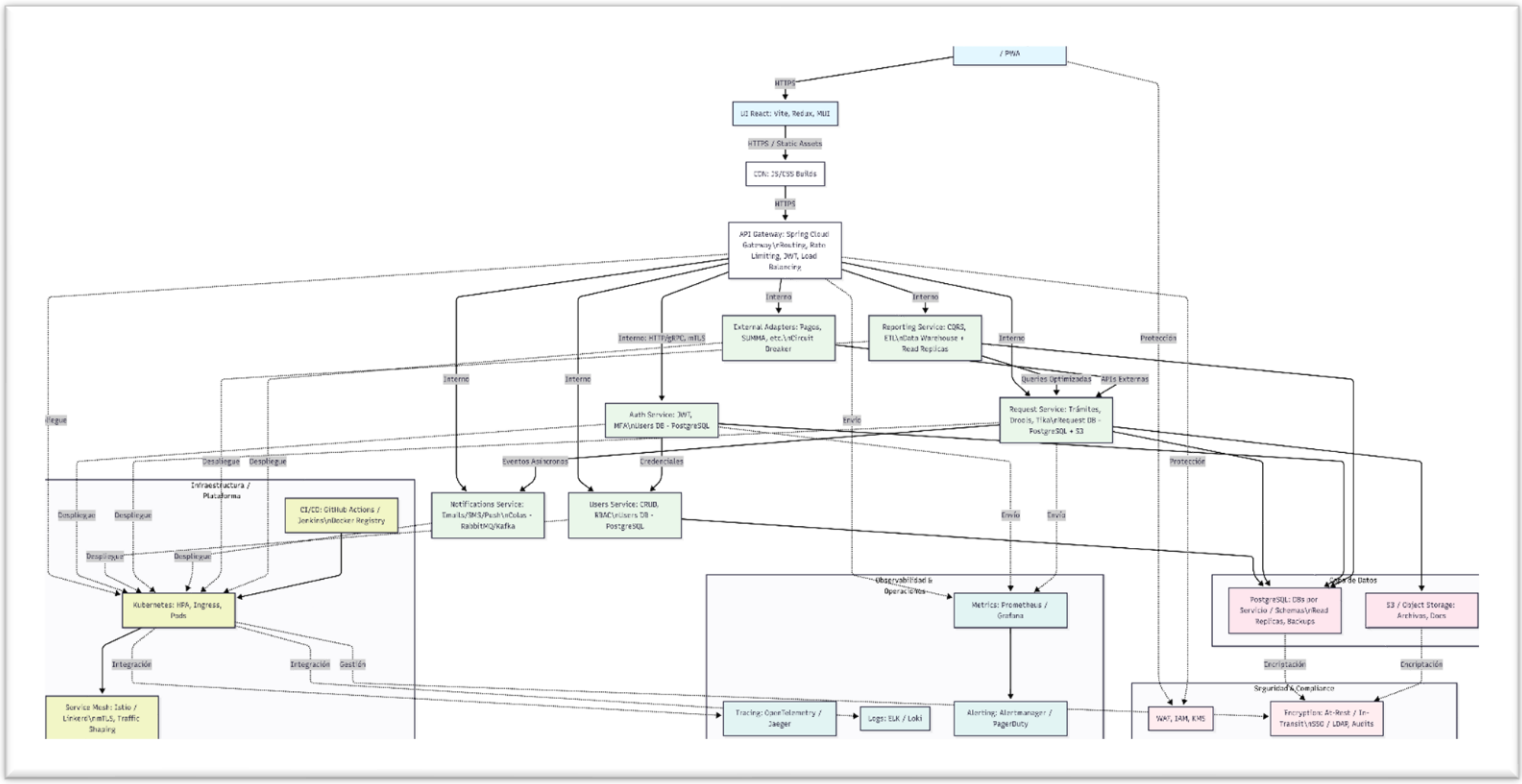
- **API de Integración**
    - implementación con CRM y SRM herramientas de la Corporación Asturias.

- **Storage**
    - PostgreSQL.multinenants, una base de datos por microservicio, backups.
    - Object Storage AWS S3 para archivos del aplicativo.

- **Infrastructure / Platform**
    - Kubernetes EKS.
    - CI/CD (GitHub Actions / Jenkins), Docker images y docker compose.

- **Observabilidad & Operaciones**
    - Metrics: Micrometer → Prometheus.
    - Logs: Loki/Grafana.
    - Alerting: Prometheus.

5. Diagrama (hacer zoom):



6. Justificación

Separación de responsabilidades por dominio de negocio: Cada servicio se centra en una funcionalidad específica del negocio, lo que permite un mayor orden, claridad y alineación con los objetivos estratégicos de la organización.

Escalabilidad independiente: Los microservicios pueden escalarse de manera autónoma según la demanda de cada módulo, optimizando el uso de recursos y evitando la sobrecarga del sistema completo.

Despliegue independiente y ciclos de desarrollo ágiles: Los equipos pueden realizar entregas continuas y rápidas sin afectar otros componentes del sistema, lo cual reduce los tiempos de respuesta al negocio y mejora la adaptabilidad frente a cambios en los requerimientos.

. **Especialización de equipos**: Cada equipo puede enfocarse en el desarrollo y mantenimiento de un microservicio en particular, aumentando la eficiencia, la calidad del software y la responsabilidad sobre su ciclo de vida.

7. Objetivos Estratégicos:

Objetivo estratégico: *Consolidar la educación virtual como un modelo sostenible, innovador y de alta calidad.*

- La adopción de una **arquitectura basada en microservicios** permite una plataforma educativa **escalable y flexible**, capaz de crecer conforme aumente el número de estudiantes, programas y se puede agregar y desplegar a las demás sedes internacionales del grupo Red SUMMA.
- El **uso de tecnologías modernas** (Spring Boot 3.0, React, Kubernetes) que aseguran que la institución mantenga una infraestructura tecnológica actualizada. Correspondiente a lo que se quiere tener y corresponda a los procesos académicos Asturias.

Objetivo estratégico: *Fortalecer la innovación y la transformación digital institucional.*

- La automatización de procesos académicos y administrativos reduce la dependencia de tareas manuales, permitiendo que las áreas de la universidad se centren en la mejora continua y mejoras de respuesta en los procesos académicos o reprocesos de los estudiantes.
- El **API Gateway y los servicios modulares** facilitan la **interoperabilidad con otras plataformas** institucionales y externas (como sistemas financieros, CRM y LMS), impulsando la integración digital.

8. Objetivos Operativos

Objetivo operativo: *Optimizar los tiempos y la eficiencia de los procesos internos.*

- La arquitectura permite **procesos automáticos y trazables** en todos los procesos académicos que se puedan automatizar, disminuyendo los errores humanos y reduciendo significativamente los tiempos de respuesta al estudiante.
- Los **microservicios especializados** (Request, Notifications, Auth) distribuyen la carga de trabajo y aseguran un rendimiento óptimo en tareas críticas.

Objetivo operativo: *Mejorar la experiencia y comunicación con los estudiantes.*

- La disponibilidad 24/7 de los servicios permite a los estudiantes acceder a sus trámites desde cualquier lugar y dispositivo.

Objetivo operativo: *Facilitar la toma de decisiones basadas en datos.*

- La inclusión de **módulos de reportes y analítica (Reporting Service)** brinda información en tiempo real sobre los procesos académicos para que los mentores y analistas consulten información del estudiante en tiempo real.

Plan de implementación y Pruebas

1. Fase de Preparación y Configuración del Entorno

Objetivo: Establecer la infraestructura tecnológica necesaria para soportar el desarrollo, despliegue y mantenimiento del sistema.

Actividades principales:

1. Configuración del repositorio central de código (GitHub o GitLab).
2. Definición del pipeline de integración continua (CI/CD) mediante Jenkins o GitHub Actions.
3. Creación de entornos separados:
    - **Desarrollo (DEV)**
    - **Pruebas (QA)**
    - **Producción (PROD)**
4. Configuración del **clúster de Kubernetes** EKS con namespaces por entorno.
5. Implementación del sistema de gestión de configuraciones (ConfigMaps y Secrets).
6. Definición de políticas de acceso y roles RBAC.

2. Fase de Desarrollo de Componentes

Objetivo: Construir e integrar los componentes definidos en la arquitectura con base a los principios de los microservicios independencia y modularidad.

Actividades principales:

1. Implementación del **Frontend (React + Vite)** con manejo de estado en Redux Toolkit y diseño responsivo con Material UI.
2. Desarrollo de los **microservicios backend** utilizando Spring Boot 3.0:
    - **JWT-AUTH Service:** autenticación, emisión y validación de tokens.

- **Users Service:** gestión de usuarios, roles y permisos.
- **Request Service:** manejo de solicitudes y procesos académicos
- **Notifications Service:** envío de correos electrónicos y notificaciones automáticas.

3. Integración del **API Gateway (Spring Cloud Gateway)** para el enrutamiento y balanceo de peticiones.
4. Configuración de las **bases de datos PostgreSQL** independientes por servicio con migraciones de Springboot (JPA)
5. Implementación de logs y monitoreo básico con Micrometer, Prometheus y Grafana

3. Fase de Integración, Pruebas y Validación

Objetivo: Garantizar la interoperabilidad y estabilidad del sistema, validando el cumplimiento de los requisitos funcionales y no funcionales.

Actividades principales:

1. Pruebas unitarias JUnit, para generar las pruebas de integración entre microservicios.
2. Validación de autenticación, autorización y control de accesos (JWT y RBAC).
3. Validación de resiliencia mediante simulación de fallos.
4. Monitoreo en tiempo real del rendimiento con Prometheus y visualización en Grafana.

4. Fase de Despliegue y Puesta en Producción

Objetivo: Desplegar la solución en el entorno productivo, garantizando disponibilidad continua y mínima afectación a los usuarios finales.

Actividades principales:

1. Despliegue de los contenedores Docker en Kubernetes.
2. Configuración de balanceo de carga con NGINX.
3. Activación de métricas y alertas operativas en Prometheus.
4. Ejecución de pruebas finales de aceptación con los usuarios clave, entrando en pruebas con usuarios como mentores y analistas de cada área y procesos.
5. Documentación de procedimientos de despliegue y mantenimiento.

5. Fase de Capacitación, Monitoreo y Mantenimiento Continuo

Objetivo: Asegurar la sostenibilidad operativa del sistema mediante capacitación, monitoreo activo y mejora continua.

Actividades principales:

1. Capacitación al personal técnico en administración de Kubernetes, CI/CD y monitoreo.
2. Entrenamiento a usuarios administrativos y académicos sobre el uso del sistema.
3. Revisión trimestral de desempeño del sistema y aplicación de mejoras continuas.
4. Registro y seguimiento de incidencias.

Estrategias de Despliegue

1. Estrategia Complementaria: Despliegue Rolling Update

Descripción:

Con el Rolling Update, las nuevas versiones se van implementando **gradualmente**, reemplazando las instancias antiguas del servicio una por una dentro del mismo entorno de producción.

Justificación:

- Apta para los **microservicios independientes** (Auth, Users, Request, Notifications), donde cada uno puede actualizarse sin afectar a los demás.
- Se alinea con el modelo de **entrega continua (CI/CD)** definido en el plan de implementación.
- Permite monitorear el comportamiento de la nueva versión y detener el proceso si se detectan errores.

Ventajas clave:

- Despliegue controlado y progresivo.
- Reducción del riesgo de errores globales.
- Compatible con Kubernetes.

2. Estrategia Principal: Despliegue Blue-Green

Descripción:

El despliegue Blue-Green consiste en mantener **dos entornos idénticos**:

- **Blue:** entorno activo (versión actual en producción).
- **Green:** entorno inactivo (versión nueva del sistema).

Una vez validada la nueva versión en el entorno Green, se redirige el tráfico al nuevo entorno mediante el balanceador de carga. Si ocurre un error, se puede revertir fácilmente al entorno anterior.

Justificación:

- Ideal para una institución educativa virtual que **no puede detener su operación** durante actualizaciones.
- Permite **actualizaciones seguras** del sistema sin interrumpir servicios críticos como matrícula, solicitudes o acceso a aulas virtuales.
- Facilita la realización de pruebas finales de usuario antes del cambio total de entorno.

Ventajas clave:

- Cero tiempos de inactividad durante despliegues.
- Capacidad de rollback inmediato ante fallos.
- Validación previa en entorno productivo simulado.

Pruebas Básicas

1. Pruebas Funcionales:

ID	Tipo de Prueba	Objetivo	Procedimiento	Resultado Esperado
F01	Autenticación de usuarios	Verificar que el servicio JWT-AUTH gestione correctamente los tokens JWT.	Enviar credenciales válidas al endpoint /auth/login y validar respuesta.	Retorna token JWT válido y tiempo de expiración correcto.
F02	Registro de usuarios	Comprobar la creación de un nuevo usuario en Users Service.	Realizar POST con datos válidos y consultar luego la base de datos.	Registro exitoso con retorno de código 201.
F03	Creación de solicitud	Validar la gestión de solicitudes en Request Service.	Enviar solicitud con datos válidos, autenticado con JWT.	La solicitud se almacena correctamente y se genera un ID único.

F04	Envío de notificación	Confirmar el envío automático de correos.	Generar evento de solicitud aprobada y observar envío de email por Notifications Service.	El correo se entrega correctamente y se registra en los logs.
F05	Integración general	Probar flujo completo: Login → Crear Solicitud → Enviar Notificación.	Ejecutar caso de uso end-to-end desde el frontend React.	Flujo completo ejecutado sin errores ni latencia excesiva.

2. Pruebas de Rendimiento

ID	Tipo de Prueba	Objetivo	Procedimiento	Resultado Esperado
R01	Prueba de carga	Evaluar el desempeño con múltiples usuarios concurrentes.	Simular 500 usuarios accediendo al sistema simultáneamente.	El sistema mantiene tiempos de respuesta menos de 3 segundos.
R02	Prueba de estrés	Determinar el punto de fallo del sistema.	Aumentar progresivamente el número de usuarios hasta que se degrade el servicio.	El sistema se degrada de forma controlada y recupera su estabilidad.
R03	Prueba de escalabilidad	Validar la respuesta del HPA (Kubernetes).	Aumentar la carga y observar el escalado automático de pods.	Los pods se incrementan

				automáticamente y el sistema se estabiliza.
R04	Prueba de latencia interna	Medir la comunicación entre microservicios.	Monitorear tiempos de respuesta del API Gateway hacia los microservicios.	Latencia < 100 ms promedio entre servicios.

4. Pruebas de Seguridad

IID	Tipo de Prueba	Objetivo	Procedimiento / Escenario	Resultado Esperado
RE01	Simulación de fallo de microservicio	Verificar reintentos automáticos.	Detener manualmente un pod del Request Service.	El sistema reintenta operación y Kubernetes lo reemplaza automáticamente.
RE02	Prueba de failover de base de datos	Validar replicación activa.	Apagar instancia principal de BD y verificar continuidad.	Redirección automática a réplica sin pérdida de datos.
RE03	Recuperación ante desastre	Evaluar restauración de backups.	Restaurar Snapchat reciente tras eliminación simulada.	Sistema recuperado en menos de 10 minutos.

Análisis Crítico de los Resultados Esperados de las Pruebas

1. Cumplimiento de los Requisitos Funcionales

Los resultados esperados de las pruebas funcionales demuestran que el sistema cumple con las operaciones definidas como esenciales: autenticación, gestión de usuarios, manejo de solicitudes, notificaciones y trazabilidad de procesos. El correcto funcionamiento de los microservicios JWT-AUTH, Users, Request y Notifications evidencia que la arquitectura mantiene la independencia modular planteada en el diseño, garantizando la integridad de los flujos de negocio.

2. Cumplimiento de los Requisitos No Funcionales de Rendimiento

Las pruebas de carga y estrés reflejan que el sistema puede mantener tiempos de respuesta inferiores a 3 segundos bajo una demanda de hasta 500 usuarios concurrentes, validando los estándares de rendimiento definidos.

3. Cumplimiento de los Requisitos de Seguridad

Los resultados de las pruebas de seguridad confirman que la arquitectura protege adecuadamente la información institucional y personal. El control de acceso mediante roles (RBAC) y la validación de tokens JWT garantizan la autenticación y autorización seguras.

5. Conclusión Global del Análisis

Categoría	Requisito Evaluado	Resultado de las Pruebas	Nivel de Cumplimiento
Funcionalidad	Correcta ejecución de procesos de negocio	Flujos completos, sin errores	Alto
Rendimiento	Respuesta < 3 s y escalado automático	Cumplido, con margen de mejora en lecturas	Medio - Alto
Seguridad	Autenticación, autorización, cifrado	Cumplido (sin vulnerabilidades críticas)	Alto

Documentación Técnica del Soporte

1. Manual de Instalación y Configuración del Sistema

Contiene las instrucciones detalladas para la instalación de cada componente:

- Configuración del entorno de desarrollo, pruebas y producción.
- Instalación de dependencias, contenedores y despliegue en Kubernetes.
- Configuración de balanceadores de carga, variables de entorno, y servicios de monitoreo.

2. Documentación de la API (Interfaces REST)

Incluye la definición de todos los endpoints disponibles en la arquitectura:

- Especificaciones de la API y como se usa en Swagger.
- Métodos, parámetros, cabeceras, ejemplos de solicitudes y respuestas.
- Políticas de autenticación (JWT) y manejo de error

3. Manual de Usuario Final (Administrativo, Docente y Estudiantil)

Describe el uso funcional del sistema desde la interfaz de usuario determinado sobre su rol y permiso:

- Ingreso y autenticación del aplicativo.
- Gestión de solicitudes, seguimiento y notificaciones.
- Roles y funcionalidades disponibles según perfil.

4. Registro de Cambios y Versiones

Documento de fechas que registra las modificaciones realizadas al sistema entre diferentes versiones:

- Nuevas funcionalidades, correcciones y mejoras.
- Versiones de software, dependencias y librerías.
- Fechas de despliegue y responsables de cada actualización.