Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa de ingeniería de sistemas
Ingeniería de Software II Grupos A y B
Laboratorio de Ingeniería de Software II Grupos A, B y C
W. Libardo Pantoja Y, Brayan Daniel Perdomo
2025.2

Ingeniería de Software II

Teoría y Laboratorio

Entregables tercer corte 2025.2

Entregables

Para este tercer corte se trabajarán los siguientes entregables:

1. Abarcar un **requisito funcional nuevo** en un microservicio: YO COMO jefe de departamento NECESITO delegar dos docentes del departamento PARA que evalúen un anteproyecto. Contexto: el sistema debe notificar a los docentes mediante un email que han sido designados como evaluadores.
2. Documentar al menos una API REST con **Swagger**.
3. Implementar la **arquitectura hexagonal** (puertos y adaptadores) con la filosofía DDD (Domain-Driven Design) a un microservicio que lo amerite (que tenga reglas en su lógica de dominio).
4. Implementar **autenticación y autorización** seguras basadas en **tokens JWT** para los usuarios y roles del sistema. Se puede hacer bajo dos esquemas. De forma manual, creando y gestionando tokens JWT directamente en la aplicación. La segunda, usando una aplicación especializada como Keycloak que gestione autenticación, autorización y emisión de tokens. En este segundo esquema, Spring Boot solo verifica y consume el JWT que emite Keycloak. Además, opcionalmente, se puede implementar una API Gateway. Cuando se usa un API Gateway en un sistema basado en microservicios con JWT, la autenticación y autorización se centralizan en el Gateway. Así, los microservicios no necesitan validar el JWT ni preocuparse por los permisos directamente: el Gateway actúa como guardián, es decir, se encarga de controlar quién puede pasar hacia los microservicios.
5. **(Opcional, +0.5 en la nota si lo hacen bien) Dockerizar** la aplicación de los microservicios y orquestarlos con **Docker-Compose**. Para uno de los microservicios se debe manejar un volumen de Docker para guardar la base de datos de manera persistente mediante Postgres o Mysql. Los otros microservicios pueden seguir manejando las bases de datos en memoria mediante H2 o Sqlite.

6. Documento final en **PDF** con la **documentación de la arquitectura** del software (el documento que se ha venido trabajando en cada corte). Se sugiere que el repositorio Github del proyecto tenga una ligera documentación del proyecto, sus requisitos, tecnología y algo de la arquitectura de tal forma que el repositorio sirva de carta de presentación cuando los estudiantes necesitan evidenciar experiencia laboral ante un sector de la industria.

A nivel de requisitos funcionales, se debe agregar dos nuevas historias y el envío de notificaciones (en color azul). A continuación, se describen los requerimientos a entregar.

Requisitos funcionales

Se deben implementar los siguientes requisitos funcionales de alto valor para el cliente:

1. **Yo como** docente **necesito** registrarme en el sistema de Gestión de Trabajos de grado **para** iniciar el flujo de un proyecto de grado, comenzando con la presentación del formato A. *Contexto*: Los datos que se deben ingresar son: nombres, apellidos, celular (dato opcional), programa al que pertenece (Ingeniería de Sistemas, Ingeniería Electrónica y Telecomunicaciones, Automática industrial, Tecnología en Telemática), email institucional y contraseña (mínimo 6 caracteres, debe tener al menos un dígito, al menos un carácter especial y al menos una mayúscula).
2. **Yo como** docente **necesito** subir un el formato A **para** comenzar el proceso de proyecto de grado. *Contexto*: el docente, una vez iniciada sesión, debe diligenciar un formulario con los datos: Título del proyecto de grado, modalidad (investigación, práctica profesional), fecha actual, director del proyecto de grado, codirector del proyecto de grado, objetivo general, objetivos específicos, archivo PDF a adjuntar. Cuando se trata de una modalidad de Práctica Profesional, el formato A debe tener al final, la carta de aceptación de la empresa. NOTA: Una vez enviado el formato A, el sistema debe enviar una notificación asíncrona al email del coordinador.
3. **Yo como** coordinador de programa **necesito** evaluar un formato A **para** aprobar, rechazar y dejar observaciones. *Contexto*: El sistema debe cargar al coordinador un listado de proyectos y su estado. Una vez evaluado el formato A, el sistema debe enviar una notificación asíncrona mediante un correo electrónico a los docentes y estudiantes implicados (se puede simular el envío con un *logger*), informando que se hizo una evaluación.
4. **Yo como** docente **necesito** subir una nueva versión del formato A cuando hubo una evaluación de rechazado **para** continuar con el proceso de proyecto de grado.
   Contexto: el requisito se parece al requisito 2, solo que lleva el conteo del número del

intento (2,3). Después de un tercer intento, el proyecto es rechazado definitivamente y el estudiante debe empezar un nuevo proyecto desde cero. NOTA: Una vez enviado el formato A (la nueva versión), el sistema debe enviar una notificación asíncrona al email del coordinador.

5. **Yo como** estudiante **necesito** entrar a la plataforma y ver el estado de mi proyecto de grado. Contexto: los estados podrían ser algo como: en primera evaluación, formato A, en segunda evaluación formato A, en tercera evaluación formato A, aceptado formato A y rechazado formato A.
6. **Yo como** docente **necesito** subir el anteproyecto **para** continuar con el proceso de proyecto de grado. *Contexto*: el docente, una vez aprobado el Formato A del proyecto, puede subir el anteproyecto para ser evaluado por jefatura. El sistema debe guardar la fecha. NOTA: Una vez enviado el anteproyecto, el sistema debe enviar una notificación asíncrona al email del jefe de departamento.
7. **Yo como** jefe de departamento **necesito** ver los anteproyectos que han sido subidos por los docentes **para** luego asignar dos evaluadores del departamento de sistemas. Contexto: la asignación de evaluadores se hará en otra historia de usuario, aquí es solamente listar.
8. **Y como** jefe de departamento **necesito** delegar dos docentes del departamento **para** que evalúen un anteproyecto. Contexto: el sistema debe notificar a los docentes mediante un email (se simula con un logger) que han sido designados como evaluadores.

Rúbrica de evaluación

Criterio	**Excelente (5 pts)**	**Bueno (4 pts)**	**Aceptable (3 pts)**	**Insuficiente (0, 1 pts)**	**Peso (%)**
Cumplimiento de los requisitos funcionales	Implementa completamente los requisitos funcionales	Los requisitos se implementan con pequeñas limitaciones o errores.	Los requisitos se implementan parcialmente o con errores funcionales importantes.	Los requisitos no se implementan o no cumplen las necesidades del cliente.	50%

Arquitectura hexagonal y DDD	Se aplica correctamente la arquitectura hexagonal, con separación clara de puertos y adaptadores, y modelado del dominio con DDD.	Se aplica la arquitectura y DDD con leves deficiencias o inconsistencias.	Se usa parcialmente o de forma incorrecta alguno de los enfoques (hexagonal o DDD).	No se identifica una arquitectura hexagonal ni modelado de dominio con DDD.	20%
Autenticación y autorización con JWT	JWT implementado de manera correcta y segura.	JWT implementado con algunos detalles mejorables.	JWT básico implementado, pero con problemas de seguridad, escalabilidad o estructura.	No se implementa JWT o es muy deficiente.	20%
Documentación de arquitectura	Documento claro, estructurado, con diagramas y detalles técnicos relevantes. Repositorio Git bien organizado y documentado	Documento y repositorio adecuados, pero con pequeños aspectos mejorables.	Documento y repositorio, con aspectos importantes a mejorar.	No hay documento de la arquitectura y repositorio Git.	10%

El documento de arquitectura debe tener estas partes:

- Portada
- Introducción breve al documento
- Historias de usuario implementadas en todas las iteraciones.
- Pantallazo de planificación de tareas: Jira, Trello.
- Escenario de calidad escalabilidad: contexto, estímulo, respuesta, medición de calidad, resultado esperado.

- Diagrama de bounded context donde se muestran los contextos delimitados y la comunicación (síncrona, asíncrona) entre ellos.
- Arquitectura y diseño de software usando el modelo C4.
- Listado de patrones de diseño implementados y una breve descripción de cómo y en qué contexto del problema se aplicaron.
- Explicación breve de cómo se implementa la arquitectura hexagonal.
- Explicación breve de cómo se implementa los mecanismos de autenticación y autorización.
- URL del video de YouTube
- URL del repositorio GIT.

PROTOCOLO DE SUSTENTACIÓN

Todos los integrantes del equipo deben participar en la sustentación. A continuación, el protocolo de sustentación que se hará de manera presencial:

1. Mostrar el tablero de tareas de la tercera iteración y la gráfica del burn down chart: **1 minuto**.
2. Mostrar en código cómo se implementa la arquitectura hexagonal de un microservicio: **1 minuto**.
3. Mostrar cómo se implementa cada uno de los patrones de diseño: **3 minutos**.
4. Mostrar cómo se implementó el esquema de autorización y autenticación: **3 minutos**.
5. Mostrar cómo se dockerizó los microservicios (si se hizo): **2 minutos**.
6. Mostrar las pruebas unitarias de algún microservicio: **1 minuto**.
7. Mostrar el software funcional evidenciando el cumplimiento de cada uno de los requisitos funcionales (Ver Listado de Requisitos Funcionales): **5 minutos**

NOTA: Los grupos que se pasen de estos tiempos, serán penalizados con -1 punto. Los puntos que no se presenten como tablero de tareas, patrones de diseño, swagger, pruebas unitarias, etcserán penalizados con **-0.5** cada uno.