Gestión de Conferencias Académicas

Universidad del Cauca

Vigilada Mineducación

Ingeniería de software II

Proyecto de clase 2024.2

Presentado por:

Carlos Santiago Balcazar
Santiago Dorado Gomez
Duber Andres Eraso Uni
Mary Luz Montenegro

Estiven Medina

Profesor:

Wilson Libardo Pantoja

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, Diciembre de 2024

CONTENIDO

Introducción

Este proyecto, tiene como objetivo principal crear una aplicación para la gestión de conferencias académicas. La organización de conferencias en el ámbito universitario enfrenta múltiples desafíos logísticos y de coordinación, especialmente en la recepción de trabajos, asignación de revisores, y manejo de inscripciones y comunicaciones. El proyecto busca automatizar estos procesos mediante una plataforma que centraliza todas las operaciones necesarias para llevar a cabo una conferencia de manera eficiente.

El problema que aborda esta aplicación surge de la necesidad de tener una herramienta especializada para gestionar eventos académicos sin depender de soluciones comerciales costosas como EasyChair. La aplicación permitirá registrar usuarios (autores, revisores, organizadores), manejar el proceso de revisión de trabajos, organizar el programa de la conferencia, entre otras funcionalidades. Asimismo, se hará énfasis en los requisitos no funcionales como la escalabilidad, usabilidad y seguridad de la aplicación.

1. Planteamiento del problema

En muchas universidades y comunidades académicas, la organización de conferencias y eventos académicos presenta desafíos significativos en términos de logística, coordinación y gestión de información. La falta de herramientas adecuadas puede llevar a procesos ineficientes, pérdida de datos y experiencias insatisfactorias tanto para los organizadores como para los participantes. Hay problemas específicos que se describen a continuación respecto a la gestión de conferencias académicas. El proceso manual de recepción y seguimiento de trabajos académicos es propenso a errores y consume mucho tiempo. Asignar trabajos a revisores adecuados, gestionar plazos y consolidar feedback puede ser un proceso caótico sin una plataforma centralizada. Crear y actualizar el programa de la conferencia, con sesiones, ponencias y horarios, puede ser complejo y propenso a errores. Gestionar inscripciones de participantes y la logística relacionada, como la confirmación de asistencia, puede ser complicado sin un sistema automatizado. La falta de un canal de comunicación eficiente puede llevar a malentendidos y falta de información crítica. La gestión manual de notificaciones y recordatorios para fechas límite, sesiones y otros eventos puede ser ineficaz. La coordinación del Programa de Ingeniería de Sistemas, a través de su coordinador, el profesor, Julio Ariel Hurtado Alegría, quien actuará como cliente, necesita que los estudiantes del curso de Ingeniería de Software II, desarrollen esta aplicación durante el periodo 2024.2.

Esta aplicación será utilizada para gestionar las conferencias que organice el departamento de sistemas en un futuro cercano. En el momento no se quiere utilizar aplicaciones comerciales como EasyChair debido a los costos que implican su licencia. A continuación se presenta una lista inicial de requisitos de esta aplicación. Estos requisitos serán capturados, depurados, detallados, priorizados y negociados con el cliente en su debido momento. El cliente elegirá el alcance del proyecto y los requisitos que se abordarán durante este periodo.

2. Thinking aloud

Se realiza el thinking aloud, obteniendo lo siguiente:

“Mi primera impresión fue que al abrir esta aplicación, me doy cuenta de que tengo un inicio de sesión y la opción también de registrarme.

En la parte de gestionar conferencia se ven unos apartados en los cuales puedo acceder a gestionar conferencia, organizadores, evaluadores, registrar autor y si es preciso salir en este caso se encuentra en gestionar conferencia donde se piden los datos como nombre de la conferencia fecha inicio fecha fin y su ubicación, finalmente teniendo un botón claro que evita confusiones para poder registrar la conferencia de forma correcta, después de darle en el botón de registrar conferencia me manda a una nueva vista en la cual ya se va a colocar la información mencionada donde en la parte de las fechas se facilita colocando el calendario para una mejor precisión a la hora de ingresar los datos y finalmente está el botón de registrar que por su nombre y la ilustración que tiene se supone fácilmente que es para finalizar el registro de la conferencia. Explorando las otras opciones ingresé al botón de gestionar artículos, aquí también encontré una vista muy similar a la anterior donde se pueden buscar y visualizar los artículos donde estos se describen por su ID título, palabras clave, el nombre del artículo o el de los autores, la conferencia y el estado de esta, al tener seleccionado un autor se puede ver que se habilitan 2 botones mas que son actualizar y asignar evaluador.

Posteriormente ingresé al botón de Gestionar evaluadores, aquí podemos ver una vista similar a la anterior pero ahora para gestionar evaluadores donde se va a poder ver la información de estos como el ID nombre y apellido y van a haber dos botones de actualizar y eliminar. Aparte de eso abajo van a aparecer otros dos botones de actualizar y registrar. En esta parte se le va a asignar un evaluador a un artículo donde vamos a poder buscarlo por una barra desplegable y seleccionar los evaluadores por en checkbox y finalizar con un botón de confirmación. En general, me parece que la aplicación es intuitiva y maneja los nombres e ilustraciones adecuadas para saber qué cosa se va a realizar.”

3. **Requisitos funcionales en forma de historias épicas, historias de usuario**

HistoriasEpicas

HistoriasUsuario

4. **Requisitos no funcionales en forma de escenarios de atributos de calidad**

Basados en los atributos de calidad clave (modificabilidad, escalabilidad y disponibilidad), se pueden plantear los siguientes escenarios:

Modificabilidad

- *Escenario:* El equipo de desarrollo necesita actualizar una funcionalidad sin afectar a todo el sistema.
- *Solución:* Se aplican microservicios debido a que estos favorecen este escenario debido a que cada servicio es independiente. Una modificación en un microservicio no impacta directamente en los demás, facilitando los cambios en el código o en la lógica de negocio sin comprometer todo el sistema.
- *Escenario:* Un nuevo microservicio debe ser agregado para gestionar una nueva funcionalidad.
- *Solución:* Permiten agregar nuevas funcionalidades sin la necesidad de modificar o intervenir en los microservicios existentes. Esto favorece la extensibilidad y el bajo acoplamiento. Por lo que se desarrolla un nuevo microservicio de manera independiente, sin acoplarlo a la lógica existente. Por tanto, se usan **principios de SOLID**, especialmente el de **Responsabilidad Única**, para que el nuevo servicio maneje exclusivamente una funcionalidad específica.

Escalabilidad

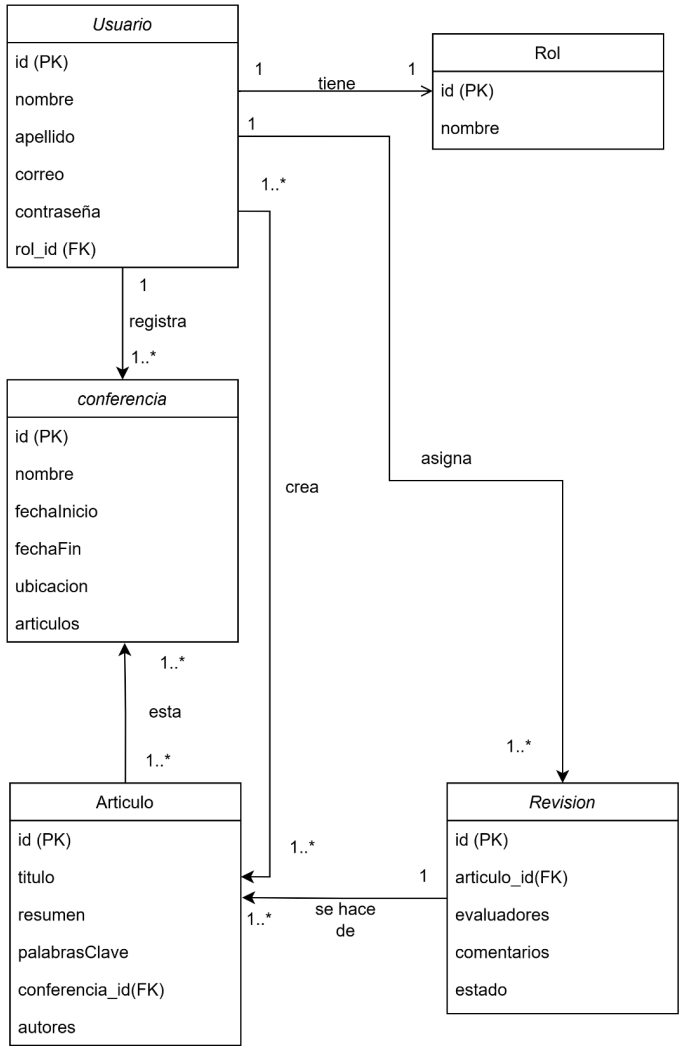
- *Escenario:* El sistema experimenta una alta demanda en un área específica, como el servicio de procesamiento de pagos.
- *Solución:* A través de microservicios, la escalabilidad se maneja a nivel de servicios individuales. Cada servicio puede ser escalado de manera independiente, agregando más instancias de ese servicio sin necesidad de escalar el resto del sistema, lo que optimiza el uso de recursos.

- *Escenario:* Se necesita una estrategia de escalabilidad horizontal (añadir más instancias de un servicio en diferentes máquinas) para soportar crecimiento.
- *Solución:* Los microservicios facilitan la escalabilidad horizontal, ya que cada microservicio puede ser distribuido y replicado de forma independiente, mejorando la capacidad del sistema para manejar mayor carga.

Disponibilidad:

- *Escenario:* El sistema debe permanecer operativo incluso si un componente falla.
- *Solución:* Los Microservicios aumentan la disponibilidad al permitir que el fallo de un servicio no afecte el funcionamiento total del sistema, puesto a que los microservicios están desacoplados y pueden ejecutarse independientemente unos de otros.
- *Escenario:* Se necesita realizar mantenimiento en un servicio sin interrumpir la funcionalidad de los otros servicios.
- *Solución:* En una arquitectura de microservicios, es posible actualizar o realizar mantenimiento en un microservicio específico sin necesidad de detener o afectar el resto de la aplicación, lo que aumenta la disponibilidad.

5. Diagrama entidad relación



6. **Arquitectura del sistema usando el modelo C4 más algunos diagramas del modelo 4+1 (los más relevantes).**

Diagrama de contexto:

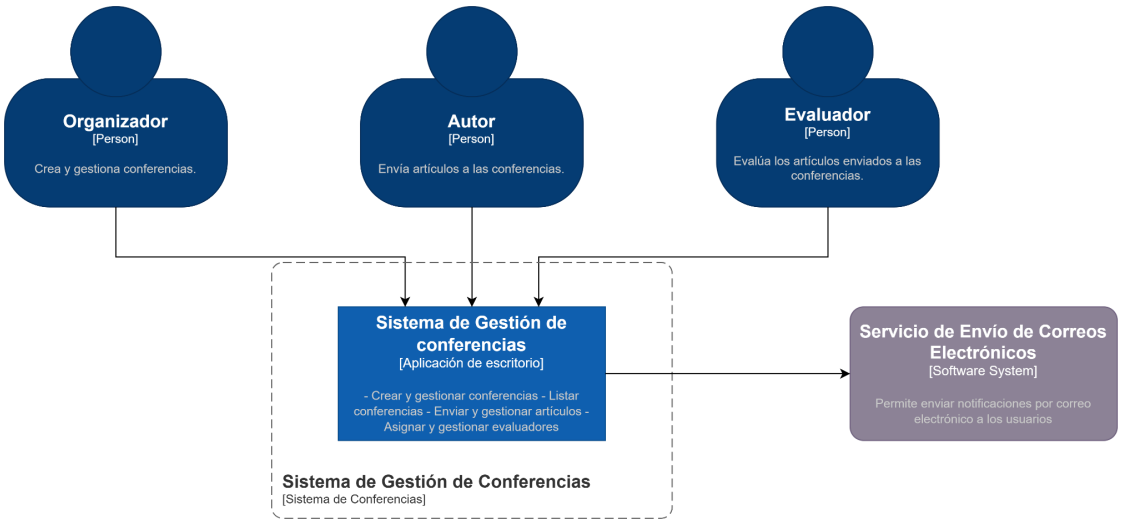


Diagrama de contenedores:

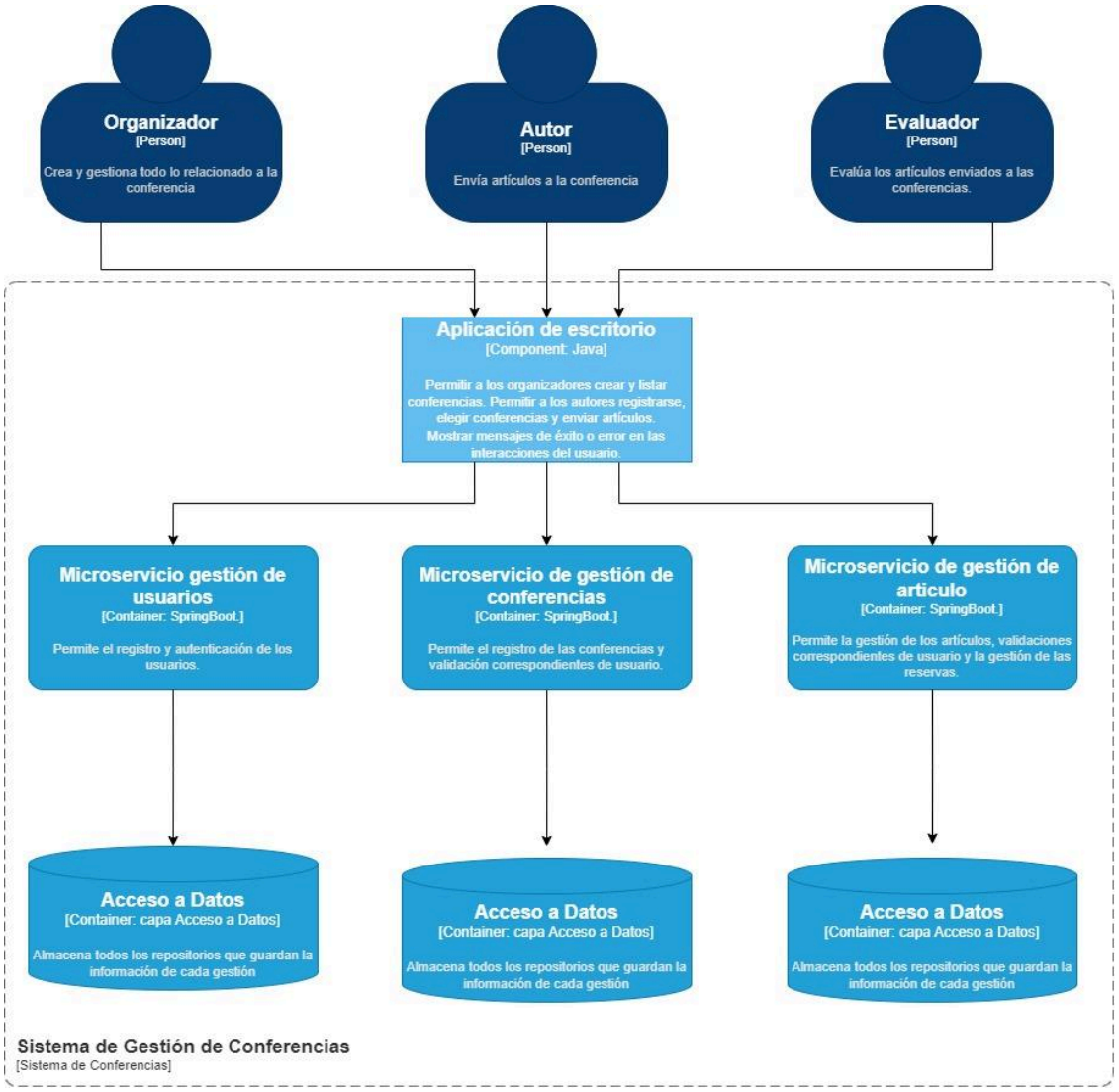
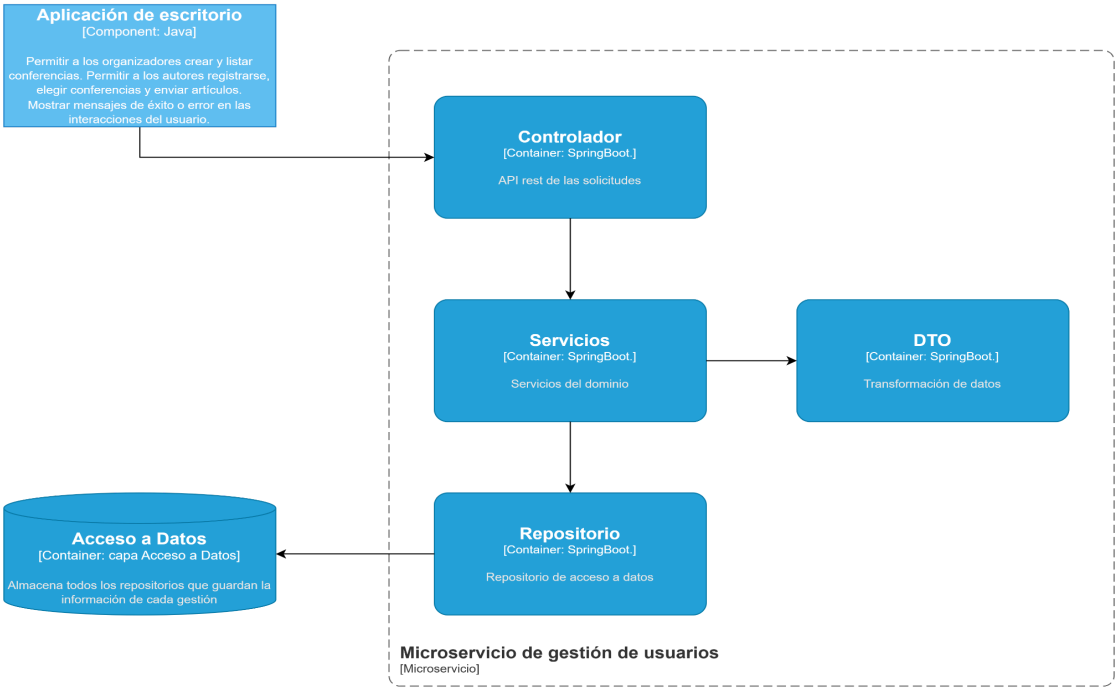
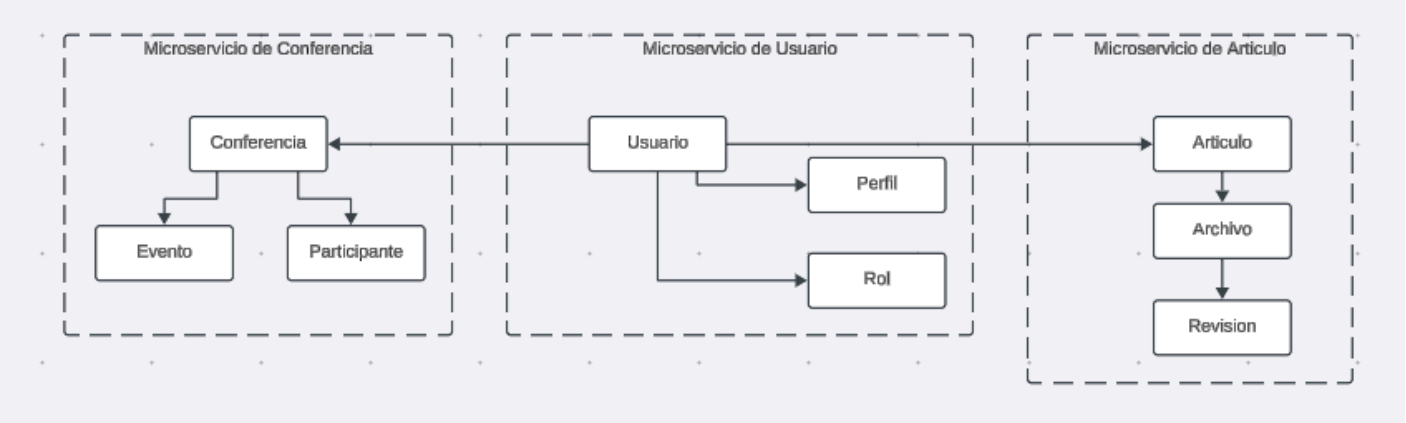


Diagrama de componentes:



7. Bounded context



8. Decisiones de arquitectura y su justificación

Tipo de aplicación:

Se ha seleccionado una aplicación basada en microservicios debido a que cada microservicio puede escalarse de manera individual, en función de la demanda de recursos específicos. Además de que se puede trabajar en microservicios independientes de forma paralela, lo que facilita el desarrollo ágil. Asimismo, cada microservicio se despliega de forma autónoma, permitiendo actualizaciones y mejoras constantes sin afectar a todo el sistema.

Estilo arquitectónico:

Este proyecto se centra principalmente en una arquitectura de comunicación de servicios basada en API. Cada microservicio expone una API REST para interactuar con los otros servicios, siguiendo principios de REST.

En este mismo sentido, cada microservicio se enfoca en un dominio específico dentro de la aplicación (por ejemplo, un microservicio para usuarios, otro para artículos y otro para conferencias.). Igualmente se utilizó un config server ya que mediante este se pudo almacenar todas las configuraciones en un único lugar, evitando la duplicación de configuraciones en cada microservicio., un eureka server para registrar los microservicios y así se indique si estos están disponibles permitiendo que los microservicios encuentren y se comuniquen con los otros microservicios sin necesidad de conocer sus puertos, ya que Eureka proporciona esta información, y gateway para asegurar que solo las solicitudes válidas lleguen a los microservicios. Es así como los microservicios cuando se inician, obtienen sus configuraciones desde el Config Server. Los microservicios se registran en el Eureka Server y así cuando un microservicio necesita comunicarse con otro, consulta el Eureka Server para obtener la ubicación del servicio requerido. Las solicitudes de los usuarios entran a través del API Gateway, que enruta las solicitudes a los microservicios correspondientes basándose en las rutas definidas.

Igualmente, se le aplicó una arquitectura hexagonal al microservicio de artículos puesto a que esta arquitectura separa el núcleo de la lógica de negocio de las preocupaciones externas como la infraestructura y las interfaces de usuario. Esto permite que, en este caso, la gestión de artículos sea más fácil de entender y mantener. En este microservicio, el núcleo de negocio (lógica de validación, etc.) no depende directamente de las tecnologías de almacenamiento (repositorios) ni de cómo se exponen las APIs. Además, los adaptadores de entrada manejan las solicitudes HTTP (controladores REST) y convierten estos datos en comandos o consultas para el núcleo de negocio y los adaptadores de salida manejan la persistencia (repositorios) y cualquier integración externa (por ejemplo, el consumir el microservicio de usuarios).

Además, se aplicaron determinados patrones en este proyecto, específicamente el patrón builder en el microservicio de conferencias puesto a que permite crear instancias de objetos que tienen muchos parámetros opcionales o requeridos. En el caso de ConferenciaEntity, donde hay varios atributos como id, nombre, fechaInicio, fechaFin, ubicación y artículos, el Builder facilita la construcción de estos objetos de manera clara y concisa. El patrón Builder proporciona una API fluida y fácil de usar para la creación de objetos, lo que facilita su integración en otros componentes del microservicio.

En este mismo sentido se aplicó el patrón decorador puesto a que este patrón en el microservicio de usuarios, debido a que este patrón es útil al añadir funcionalidades como roles, permisos, o comportamientos específicos, facilitando la separación de preocupaciones, haciendo que cada demorador se encargue de una responsabilidad específica.

Además, se agregó el patrón Observer en el front-end cuando se crean conferencias y artículos, puesto a que este es un patrón de diseño que permite a un objeto

notificar a otros objetos sobre cambios en su estado sin que estos estén acoplados entre sí. Cuando se crea una conferencia o artículo, todos los observadores interesados pueden ser notificados automáticamente y actualizar su estado en consecuencia, asegurando que la interfaz de usuario refleje siempre el estado más reciente de los datos.

Tecnologías:

Entorno de desarrollo: Visual Studio Code, NetBeans