Justificación del Diseño e Implementación de la Lógica de la Aplicación

1. Estructura de las Entidades y Relaciones

En el diseño de la aplicación, se establecieron las siguientes entidades principales: **Application**, **Version**, **TestCycle**, y **MetricEntity**. La estructura de estas entidades y sus relaciones se basa en la necesidad de representar de manera efectiva un sistema de registro de métricas de prueba.

- **Application**: Representa una aplicación específica que se va a gestionar. Cada aplicación puede tener múltiples versiones asociadas.
- **Version**: Representa una versión particular de una aplicación. Cada versión puede tener múltiples ciclos de prueba asociados.
- **TestCycle**: Representa un ciclo de prueba que se ejecuta para una versión específica. Cada ciclo de prueba puede tener múltiples métricas asociadas.
- **MetricEntity**: Representa una métrica individual que mide un aspecto del ciclo de prueba, como el número de casos ejecutados o el tiempo total de ejecución.

Esta estructura permite una organización jerárquica clara y un almacenamiento eficiente de datos, facilitando las operaciones CRUD y la consulta de información específica.

2. Decisiones de Diseño en la Lógica de Creación

2.1. Creación de Aplicaciones y Versiones

El proceso de creación de una nueva aplicación y sus versiones asociadas se diseñó teniendo en cuenta la integridad referencial y la consistencia de los datos.

- **Validación de existencia**: Antes de crear una nueva aplicación, se verifica si ya existe una con el mismo nombre para evitar duplicados. Esta lógica está representada en el diagrama de clases, donde el método `create` del servicio de aplicaciones (`ApplicationService`) realiza una consulta al repositorio para verificar la existencia.
- **Creación y asociación**: Cuando se crea una aplicación, se instancian objetos de tipo `Application` y se persisten utilizando el repositorio de aplicaciones. Para las versiones, se aplica una lógica similar, asegurando que cada versión esté asociada a una aplicación existente, preservando así la integridad referencial.

2.2. Creación de Ciclos de Prueba y Métricas

La creación de ciclos de prueba y métricas sigue una lógica centrada en la relación jerárquica de las entidades:

- **Asociación directa**: Al crear un ciclo de prueba, se asocia directamente a una versión específica de la aplicación, que es un requisito para mantener la integridad y la coherencia en el sistema. El diagrama de objetos ilustra cómo cada `TestCycle` está vinculado a una instancia de `Version`, que a su vez está vinculada a una instancia de `Application`.
- **Configuración de métricas**: Las métricas se configuran durante la creación del ciclo de prueba, lo que asegura que todas las métricas relevantes estén asociadas al ciclo correspondiente en el momento de su creación. Esto se refleja en el código donde el ciclo de prueba (`TestCycle`) se guarda junto con sus métricas asociadas, utilizando cascadas y relaciones de persistencia adecuadas.

2.3. Cálculo de Métricas Adicionales

- **Métricas calculadas**: Se diseñó una lógica adicional para calcular métricas específicas (como la tasa de fallos de pruebas y el tiempo promedio por caso de prueba) después de la creación de un ciclo de prueba. Estas métricas se calculan en función de otras métricas existentes, lo que permite un análisis más profundo sin requerir datos adicionales de entrada del usuario. Esta decisión se basa en la necesidad de proporcionar información detallada y significativa de manera automática.

3. Diagramas de Soporte y Lógica Implementada

Los diagramas proporcionados, como el diagrama de clases, el diagrama de secuencia y el diagrama de componentes, justifican las decisiones de diseño y la lógica implementada:

- **Diagrama de Clases**: Muestra la estructura de las entidades y cómo se relacionan entre sí. Justifica el uso de relaciones `OneToMany` y `ManyToOne` para manejar las asociaciones entre aplicaciones, versiones, ciclos de prueba y métricas.
- **Diagrama de Secuencia**: Representa la interacción dinámica entre los componentes cuando se realiza una operación de creación. Muestra cómo el flujo de datos y la invocación de métodos se producen de manera secuencial, garantizando la consistencia de los datos y el manejo adecuado de excepciones.
- **Diagrama de Componentes**: Ilustra los diferentes servicios y repositorios involucrados en la aplicación, destacando la separación de responsabilidades y el uso de interfaces para la abstracción. Esto apoya la decisión de tener servicios específicos como `ApplicationService`, `VersionService` y `TestCycleService` para gestionar las operaciones CRUD correspondientes.

4. Decisiones Adicionales

- **Uso de Lombok**: Para reducir la verbosidad del código y mejorar la mantenibilidad, se utilizó Lombok para generar automáticamente getters, setters y constructores. Esto se justifica en el diagrama de clases, donde se observa una simplificación de las clases de entidad.

- **Validaciones y Manejo de Excepciones**: Se implementaron validaciones para asegurar que todos los datos necesarios estén presentes y correctos antes de realizar cualquier operación de persistencia. Además, se manejan excepciones específicas para proporcionar retroalimentación clara en caso de errores.

Conclusión

El diseño de la lógica de la aplicación y la implementación están basados en principios de buen diseño de software, tales como la separación de responsabilidades, la integridad de los datos y la facilidad de mantenimiento. Cada decisión tomada en el diseño del sistema está respaldada por una necesidad funcional o técnica, y los diagramas proporcionados ayudan a visualizar y justificar estas decisiones de manera efectiva.