Parcial 3

Universidad del Cauca

Vigilada Mineducación

Ingeniería de Software II

Presentado por:

Valentina Fernandez Guerrero

Jose David Chilito Cometa

Juan Esteban Yepez Rodriguez

Profesor:

Julio Ariel Hurtado Alegria

Ricardo Zambrano

Universidad del Cauca

Facultad de Ingeniería electrónica y telecomunicaciones

Ingeniería en Sistemas

Popayán, Noviembre del 2023

Punto 1:

i) Establezca la pila del producto (Backlog) y describa los requisitos funcionales prioritarios (Primer Sprint) a través de historias de usuario o casos de uso (utilice los templates usados en Ingeniería de Software I)

Pila del producto (Backlog)

La pila del producto está compuesta por las siguientes historias de usuario:

- HU1: Agregar acción
- HU2: Eliminar acción
- HU3: Visualizar acción
- HU4: Notificar cambios de las acciones

Requisitos funcionales prioritarios (Primer Sprint)

Los requisitos funcionales prioritarios para el primer sprint son los siguientes:

Casos de uso:

Nombre	Agregar Acción
Autor	Jose David Chilito Cometa, Valentina Fernandez Guerrero
Objetivo	Permitir a un usuario agregar una nueva acción a la bolsa de valores.
Actores	Usuario
Precondiciones	● El usuario debe estar registrado en el sistema. ● La información de la acción debe ser válida.
Descripción	El usuario ingresa la información de la acción, como el nombre,precio actual, umbral inferior y umbral superior.. El sistema agrega la acción a la bolsa de valores.
Flujo principal	**1. El usuario ingresa la información de la acción.** **2. El sistema verifica que la información sea válida.** **3. El sistema agrega la acción a la bolsa de valores.** **4. El sistema muestra un mensaje de confirmación.**
Flujos alternativos	Si la información de la acción no es válida, el sistema muestra un mensaje de error.
Postcondiciones	La acción se agrega a la bolsa de valores

Nombre	Eliminar Acción
Autor	Jose David Chilito Cometa, Valentina Fernandez Guerrero
Objetivo	Permitir a un usuario eliminar una acción existente de la bolsa de valores
Actores	Usuario
Precondiciones	● El usuario debe estar registrado en el sistema. ● El usuario debe ser el propietario de la acción.

Descripción	El usuario selecciona la acción que desea eliminar. El sistema elimina la acción de la bolsa de valores
Flujo principal	**1. El usuario selecciona la acción que desea eliminar.** **2. El sistema verifica que el usuario sea el propietario de la acción.** **3. El sistema elimina la acción de la bolsa de valores.** **4. El sistema muestra un mensaje de confirmación.**
Flujos alternativos	Si el usuario no es el propietario de la acción, el sistema muestra un mensaje de error.
Postcondiciones	La acción se elimina de la bolsa de valores.

Nombre	Visualizar Acción
Autor	Jose David Chilito Cometa, Valentina Fernandez Guerrero
Objetivo	Permitir a un usuario visualizar la información de una acción
Actores	Usuario
Precondiciones	El usuario debe estar registrado en el sistema
Descripción	El usuario ingresa a la opción de visualizar acciones. El sistema muestra una lista de las acciones. El usuario puede ver información sobre cada acción.
Flujo principal	**El usuario ingresa para buscar las acciones.** **El sistema muestra una lista de las acciones.** **El usuario puede ver información sobre cada acción.**
Flujos alternativos	Si no se encuentran acciones registradas, el sistema muestra un mensaje de error.
Postcondiciones	El usuario tiene una visión general de las acciones.

Nombre	Notificar cambios de las acciones
Autor	Jose David Chilito Cometa, Valentina Fernandez Guerrero
Objetivo	Permitir al servidor notificar a los usuarios de los cambios en las acciones.
Actores	Servidor
Precondiciones	El servidor está configurado para notificar cambios de las acciones.
Descripción	El servidor detecta un cambio en una acción. El servidor genera una notificación. El servidor envía la notificación a los usuarios que están suscritos a los cambios de la acción.
Flujo principal	**1. El servidor detecta un cambio en una acción.** **2. El servidor genera una notificación.** **3. El servidor envía la notificación a los usuarios que están suscritos a los cambios de la acción.**
Flujos alternativos	Si no hay usuarios suscritos a los cambios de la acción, el servidor no envía la notificación.
Postcondiciones	Los usuarios que están suscritos a los cambios de la acción reciben una notificación sobre el cambio.

ii) Establezca las cualidades del sistema: describa dos escenarios de calidad: uno para escalabilidad y otro para modificabilidad, considerando los requisitos no funcionales indicados.

Cualidades del Sistema

Escenario de calidad para escalabilidad:

- El sistema debe ser capaz de manejar 100 usuarios simultáneos (Municipio).
- Se prevé escalar a nivel departamental (900 usuarios simultáneos) y nacional (3,000 usuarios simultáneos).

Escenario:

En una ocasión excepcional, como la introducción del sistema para el monitoreo de acciones en un mercado agrícola a nivel nacional, anticipamos un notable incremento en la cantidad de usuarios concurrentes. Imaginemos que durante este evento específico, la cantidad de usuarios simultáneos podría llegar a los 3,000.

Esta situación refleja la necesidad de que el sistema sea capaz de manejar eficazmente la carga adicional generada por eventos de gran envergadura, como la implementación a nivel nacional en el mercado agrícola. Este escenario destaca la importancia de una escalabilidad robusta para asegurar un rendimiento óptimo incluso durante picos de demanda significativos.

Comportamiento Esperado:

- El sistema debe ser capaz de escalar horizontalmente, distribuyendo la carga entre varios servidores, para manejar el aumento proyectado de usuarios simultáneos.
- Debería implementarse una gestión eficiente de recursos para garantizar un rendimiento óptimo incluso durante picos de carga.
- La arquitectura del sistema debe ser modular y flexible, permitiendo ajustes en tiempo real para adaptarse a cambios en la demanda sin afectar la disponibilidad o la velocidad de respuesta.

Al integrar Docker y RabbitMQ, nuestro diseño se alinea de manera efectiva con los requisitos de escalabilidad, permitiendo la gestión eficiente de recursos y la comunicación asincrónica robusta entre los diferentes módulos del sistema.

Escenario de calidad para modificabilidad:

- El sistema debe ser fácil de modificar.
- Cambios de base de datos en 1 semana con un esfuerzo de 2 personas-mes.
- Cambios en gráficos de cambio de acciones en 1 semana con 0.5 personas-mes.

Escenario:

Supongamos que se requiere una modificación sustancial en la estructura de la base de datos utilizada para almacenar la información de acciones. Esta modificación es necesaria para mejorar la eficiencia y permitir un manejo más

efectivo de grandes volúmenes de datos durante eventos significativos en el mercado.

Comportamiento Esperado:

- La arquitectura del sistema debería permitir una fácil extensión de la lógica de presentación para acomodar nuevas métricas en los gráficos.
- Se espera que la implementación de estos cambios en los gráficos de cambio de acciones se realice dentro del plazo especificado de 1 semana, con un esfuerzo total de 0.5 personas-mes.
- La modularidad del código facilitará la identificación y modificación de componentes específicos sin afectar la integridad del sistema.

iii) Tome decisiones acerca de las tácticas de escalabilidad y modificabilidad a utilizar. Decida al menos dos por cada cualidad, indique qué patrones de diseño arquitectónicos serían recomendables.

Tácticas de escalabilidad

1. Load Balancing: Distribuye la carga de trabajo entre varios servidores, lo que ayuda a mejorar el rendimiento y la disponibilidad del sistema. Esto mejora la respuesta del sistema y facilita la escalabilidad horizontal.
2. Performance Retention:Mantiene un rendimiento constante. Esto puede incluir el uso de almacenamiento en caché y optimización de consultas para garantizar respuestas rápidas.

Tácticas de Modificabilidad:

1. Increase cohesion: Aumenta la cohesión de los módulos, lo que ayuda a que los módulos sean más fáciles de entender y modificar. Esta táctica se puede aplicar al diseño de los módulos, asegurándose de que cada módulo tenga un solo propósito.
2. Reduce Coupling: Minimiza las dependencias entre los componentes de la capa de servicios mediante el uso de interfaces claras y contratos bien definidos. Reducir el acoplamiento permite realizar modificaciones en un componente sin afectar a otros.

Algunos patrones de diseño arquitectónico que se podrían utilizar son:

Patrón de Capas: Divide la aplicación en capas (presentación, lógica de negocio, acceso a datos) para organizar y separar las responsabilidades. Facilita la escalabilidad y la modularidad al separar las diferentes preocupaciones de la aplicación. Spring Boot se adapta bien a este enfoque, ya que se puede organizar los servicios en capas fácilmente.

Patrón Cliente-Servidor: Divide el sistema en dos componentes principales: el cliente y el servidor. El cliente es responsable de la interacción con el usuario, mientras que el servidor es responsable de la lógica de negocio y el acceso a datos. Esto permite escalar el sistema de forma independiente, ya sea agregando más clientes o más servidores. Los componentes individuales se pueden separar en el

cliente y el servidor. Esto facilita la comprensión del sistema y la identificación de los componentes que deben modificarse.

Patrón de Microservicios: Permite escalar y desplegar servicios de manera independiente, facilitando el crecimiento del sistema y la introducción de nuevas funcionalidades. Además, ayuda a cumplir con el requisito de fácil modificación.

Patrón de Repositorio: Utiliza un patrón de repositorio para abstraer el acceso a la base de datos y permitir una fácil sustitución de la tecnología de almacenamiento de datos. Proporciona modularidad al separar la lógica de acceso a datos de la lógica de negocio principal, lo que facilita la sustitución o actualización de la base de datos.

Patrón de Actualización de Datos Asíncrona: Utiliza técnicas asíncronas para actualizar los datos, especialmente en situaciones donde se espera una alta concurrencia. Spring Boot y Spring Cloud Events pueden ser útiles para implementar esto. Mejora la escalabilidad al permitir que las operaciones de actualización de datos no bloqueen las operaciones principales y mejora la modularidad al reducir la dependencia directa entre los servicios.

v) Tome decisiones acerca de las perspectivas a usar para describir la arquitectura y escoja un patrón arquitectónico para cada una de ellas.

Dentro del enfoque ADD (Diseño Impulsado por Atributos), las elecciones relacionadas con las perspectivas y los patrones arquitectónicos deben fundamentarse en los atributos clave identificados para el sistema de seguimiento del mercado de valores.

En otras palabras, al tomar decisiones sobre cómo diseñar la arquitectura del sistema, se debe tener en cuenta y priorizar aspectos clave o características específicas del sistema que son críticas para su éxito. Estos atributos clave pueden incluir, por ejemplo, rendimiento, seguridad, escalabilidad, y otros requisitos esenciales identificados durante el proceso de diseño.

1) Perspectiva de Cliente-Servidor:
    - Patrón Arquitectónico: Hexagonal
    - Justificación (Atributos Clave): Para abordar el atributo de rendimiento y la necesidad de distribuir eficientemente la carga, se opta por la arquitectura hexagonal. Los puertos y adaptadores permiten una clara separación de las preocupaciones y facilitan la distribución de las solicitudes entre los clientes y el servidor, asegurando una respuesta eficiente a las solicitudes de seguimiento del mercado en tiempo real.
2) Perspectiva de Capas:
    - Patrón Arquitectónico:Hexagonal
    - Justificación (Atributos Clave): La modularidad y la capacidad de escalabilidad son esenciales para manejar el crecimiento del sistema. La arquitectura hexagonal proporciona una estructura organizativa que facilita la organización de las responsabilidades del sistema, permitiendo una escalabilidad más sencilla y una gestión eficiente de los cambios.
3) Perspectiva de Seguridad:
    - Patrón Arquitectónico: Hexagonal con Capas de Seguridad.
    - Justificación (Atributos Clave): La seguridad es crítica, especialmente para proteger la información financiera. La arquitectura hexagonal, combinada con capas de seguridad, garantiza la protección de datos

en todas las capas del sistema, manteniendo una clara separación de las responsabilidades y asegurando la integridad y confidencialidad de los datos financieros.

4) Perspectiva de Escalabilidad:
    - Patrón Arquitectónico: Hexagonal con Escalabilidad Horizontal.
    - Justificación (Atributos Clave): Dada la proyección de crecimiento del sistema, la escalabilidad horizontal se selecciona para permitir la adición de más servidores y distribuir eficientemente la carga, asegurando un rendimiento óptimo. La arquitectura hexagonal facilita esta escalabilidad al proporcionar una clara separación de las interfaces y la lógica del negocio.

La elección del patrón hexagonal se alinea de manera coherente con las características clave y requisitos del sistema de seguimiento del mercado de valores. La arquitectura hexagonal, mediante puertos y adaptadores, proporciona una clara separación de las preocupaciones y facilita la distribución eficiente de las solicitudes. Esta elección se basa en los atributos clave identificados, como rendimiento, modularidad, escalabilidad y seguridad.

A continuación, se proporciona las razones por la cual este patrón fue escogido:

1) Separación de responsabilidades financieras: La arquitectura hexagonal facilita la segregación de responsabilidades, asegurando que las interfaces del sistema estén claramente definidas. Los puertos y adaptadores permiten una distribución eficiente de las solicitudes, cumpliendo con las regulaciones financieras y garantizando una ejecución transparente de las operaciones.
2) Modularidad y escalabilidad: La modularidad es esencial para manejar el crecimiento del sistema, y la arquitectura hexagonal proporciona una estructura organizativa que facilita la escalabilidad. Cada puerto y adaptador puede escalar de manera independiente según sea necesario, permitiendo ajustes específicos en cada parte del sistema.
3) Facilidad en el desarrollo y mantenimiento:La seguridad es crítica en aplicaciones financieras, y la arquitectura hexagonal, combinada con capas de seguridad, asegura la protección de datos en todas las capas. La clara separación de las interfaces facilita la implementación de medidas de seguridad específicas en cada capa, garantizando la confidencialidad e integridad de la información financiera.
4) Adaptabilidad a los cambios en la bolsa de valores: Las bolsas de valores son entornos dinámicos, y la arquitectura hexagonal permite una adaptabilidad eficiente a cambios en el mercado. Cada adaptador puede ajustarse independientemente para cumplir con nuevas regulaciones o incorporar nuevas funcionalidades requeridas por la bolsa, asegurando una respuesta ágil a las dinámicas del mercado.

vi) Describa la arquitectura usando los cuatro niveles del modelo C4

Nivel 1: Contexto del sistema

Muestra las principales entidades que interactúan con la aplicación de seguimiento del mercado de valores. Las entidades principales son:

- Usuarios: Los usuarios son las personas que utilizan la aplicación para seguir los precios de las acciones y cambiar precios de los umbrales.

- Bolsa de valores: La bolsa de valores proporciona los datos de precios de las acciones y si alguna acción tiene un cambio se le notifica al usuario.

También muestra que la aplicación se comunica con la bolsa de valores a través de un servicio web.

Nivel 2: Contenedores

Descompone el sistema en contenedores. Un contenedor es una unidad de software que contiene uno o más componentes. Los contenedores del sistema de seguimiento del mercado de valores son:

- Interfaz de usuario: El contenedor de la interfaz de usuario proporciona una interfaz para que los usuarios interactúen con la aplicación.
- Capa de servicios: La capa de servicios proporciona las funcionalidades de la aplicación, como la adición de nuevas acciones, la eliminación de acciones existentes y la configuración de umbrales de precio.

Nivel 3: Componentes

Descompone los contenedores en componentes. Un componente es una unidad de software que implementa una funcionalidad específica. Los componentes del sistema de seguimiento del mercado de valores son:

- Componente de interfaz de usuario: El componente de interfaz de usuario proporciona la interfaz gráfica de usuario para la aplicación. Se implementará utilizando la biblioteca de interfaz de usuario web Postman.
- Componente de servicios: El componente de servicios implementa las funcionalidades de la aplicación, como la adición de nuevas acciones, la eliminación de acciones existentes y la configuración de umbrales de precio. Se implementará utilizando el marco de microservicios Spring Boot.
- Componente de datos: El componente de datos almacena los datos de las acciones, como el nombre, el precio actual, el precio anterior, el umbral inferior y el umbral superior. Se implementará utilizando una base local.

Pruebas de los servicios a través de Postman y Swagger:

- Para swagger se corre el programa y se ingresa al siguiente link.

  http://localhost:8080/swagger-ui/index.html
- Para Postman se adjunta un archivo JSON que se encuentra en la carpeta.