Parcial Final

Universidad del Cauca

Vigilada Mineducación

Ingeniería de Software II

Presentado por:

Katherin Alexandra Zuñiga Morales

Jeferson Castaño Ossa

David Santiago Fernández Dejoy

David Santiago Girón Muñoz

Profesor:

Julio Ariel Hurtado Alegria

Ricardo Zambrano

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, Noviembre de 2023

CONTENIDO

Segundo Parcial

1. Requisitos del Sistema y Análisis de la Arquitectura

1.1. Pila del producto y requisitos funcionales prioritarios

1.1.1. Pila del producto (Backlog)

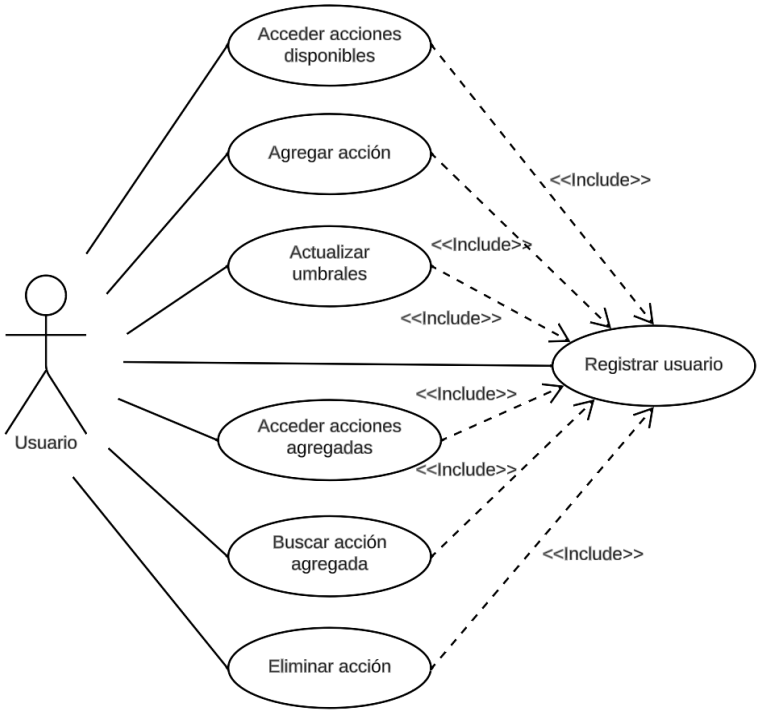
El producto Backlog es uno de los aspectos fundamentales a tener en cuenta a lo largo del proceso de desarrollo de un sistema, listando por prioridad sus funcionalidades, mejoras y correcciones que se desean implementar. Además, es dinámico y se ajusta continuamente a medida que se adquiere un mayor entendimiento del proyecto y se evoluciona a lo largo del tiempo.

Con esto en mente, se identifica un primer acercamiento al product backlog del sistema de seguimiento del mercado de valores, el cual es el siguiente:

1. Registro de usuario: Es fundamental tener noción de cada uno de los usuarios dentro del sistema para poder realizar las demás funcionalidades que se brindan.
2. Despliegue de información de las acciones de la bolsa de valores: Se desea tener al alcance la información de todas las acciones que el usuario tiene disponibilidad para agregar y seguir sus precios.
3. Agregar acción: El usuario tiene la posibilidad de llevar un seguimiento de precios de las acciones que desee agregar, estableciendo unos umbrales inferior y superior iniciales.
4. Modificar umbrales: Si se desea, el usuario puede actualizar los umbrales inicialmente establecidos para el seguimiento de precios.
5. Eliminar acción: Se desea que el usuario tenga la posibilidad de eliminar cualquiera de las acciones que haya agregado con anterioridad.
6. Despliegue de información de las acciones agregadas: El usuario puede consultar las acciones que ha agregado con su información detallada.
7. Consultar acción agregada: Para una mayor agilidad en la búsqueda de una acción, puede consultar directamente una de estas para ver si sus precios han cambiado o no.

1.1.2. Casos de uso

Diagrama de casos de uso del sistema:



Documentación Casos de Uso:

CU 01	**Registrar Usuario**
Versión	1.0 (11-11-2023)
Actor	Usuario
Dependencias	● Contar con un identificador
Precondición	El usuario debe contar con su número de identificación único para poder registrarse como un nuevo usuario.
Descripción	Permite a un usuario registrarse en el sistema StockMarket para acceder a sus funcionalidades.
Secuencia normal de los eventos	1. El caso de uso inicia cuando el usuario está interesado en registrarse dentro del sistema y selecciona la opción “Registrarse”. 2. El usuario ingresa su identificación con la cual se desea registrar. (E-1). 3. Se ingresa al sistema como un usuario nuevo. 4. Fin del caso de uso.

Postcondición	El usuario queda registrado exitosamente y se guarda su información en la base de datos.
Excepciones	E-1: 2.1 El número de identificación ya se encuentra registrado dentro del sistema. 2.1.1 Se informa al usuario y no se permite registrar al usuario dentro del sistema. 2.1.3 Fin del caso de uso.
Comentarios	

CU 02	**Acceder acciones disponibles**
Versión	1.0 (11-11-2023)
Actor	Usuario
Dependencias	● No aplica
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	Permite visualizar todas las acciones de la bolsa de valores al usuario.
Secuencia normal de los eventos	1. El caso de uso inicia en el momento en que el usuario desea visualizar todas las acciones de la bolsa de valores. 2. El usuario se dirige al apartado “Acciones” 3. El sistema muestra todas las acciones de las que se tiene registro. (E-1) 4. Fin del caso de uso.
Postcondición	Información de todas las acciones disponibles en pantalla.
Excepciones	E-1: 3.1. No se tiene registro de ninguna acción. 3.1.1 Se muestra el mensaje de indisponibilidad de acciones. 3.1.2 Fin del caso de uso.
Comentarios	

CU 03	**Agregar Acción**
Versión	1.0 (11-11-2023)
Actor	Usuario

Dependencias	● No aplica.
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	El usuario agrega una nueva acción a la cual desea hacer un seguimiento de su precio.
Secuencia normal de los eventos	1. El caso de uso inicia cuando el usuario desea agregar una primera acción o una nueva. 2. El usuario se dirige a la opción de “Agregar acción”. 3. El usuario ingresa la información de la acción a agregar en los campos disponibles (Id acción, umbral inferior, umbral superior). (S-1). 4. El sistema valida si todos los campos obligatorios están llenos. (E-1) 5. Se habilita la opción “Agregar”. 6. El usuario selecciona la opción “Agregar”. (E-2) (E-3) 7. Fin del caso de uso.
Postcondición	La información de la acción agregada por el usuario queda almacenada en la base de datos.
Sub-flujos	S-1: 3.1 El usuario no desea continuar con el proceso. 3.1.1 Selecciona la opción “Cancelar”. 3.1.2 Fin del caso de uso.
Excepciones	E-1: 4.1 No todos los campos están llenos. 4.1.1 Los campos obligatorios faltantes se resaltan en rojo y el botón “Agregar” permanece deshabilitado. 4.1.2 Volver al paso 3. E-2: 6.1 La acción ingresada no existe. 6.1.1 Se informa al usuario. 6.1.2 Volver al paso 3. E-3: 6.2 La acción ya ha sido agregada por el usuario con anterioridad. 6.2.1 Se informa al usuario. 6.2.2 Volver al paso 3.
Comentarios	

CU 04	**Actualizar Umbrales**
Versión	1.0 (11-11-2023)
Actor	Usuario

Dependencias	● No aplica.
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	Permite actualizar los umbrales de precio para las notificaciones en el cambio de precio de la acción.
Secuencia normal de los eventos	1. El caso de uso inicia cuando el usuario desea modificar los umbrales de una de las acciones que sigue. 2. El usuario se dirige a la opción de “Actualizar umbrales”. 3. El usuario ingresa el Id de la acción a la cual modificar los umbrales. 4. El usuario selecciona la opción “Modificar umbral inferior”(S-1), “Modificar umbral superior”(S-2) o “Modificar ambos umbrales”(S-3). 5. El sistema valida si todos los campos obligatorios están llenos. (E-1) 6. Se habilita la opción “Guardar”. 7. El usuario selecciona la opción “Guardar”. (E-2) 8. Fin del caso de uso.
Postcondición	Los umbrales de precio quedan establecidos en los nuevos valores y guardados en la base de datos.
Sub-flujos	S-1: 4.1 Modificar umbral inferior 4.1.1 Se habilita un único campo para modificar el umbral inferior. 4.1.2 El usuario ingresa el valor del umbral inferior. 4.1.3 Ir al paso 5. S-2: 4.2 Modificar umbral superior 4.2.1 Se habilita un único campo para modificar el umbral superior. 4.2.2 El usuario ingresa el valor del umbral superior. 4.2.3 Ir al paso 5. S-2: 4.3 Modificar ambos umbrales 4.3.1 Se habilitan dos campos para modificar los umbral inferior y superior. 4.3.2 El usuario ingresa el valor del umbral inferior y el umbral superior. 4.3.3 Regresar al paso 5.
Excepciones	E-1: 5.1 No todos los campos están llenos. 5.1.1 Los campos obligatorios faltantes se resaltan en rojo y el botón “Guardar” permanece deshabilitado. 5.1.2 Volver al paso 3. E-2: 7.1 La acción de los umbrales no existe. 7.1.1 Se informa al usuario. 7.1.2 Volver al paso 3.
Comentarios	

CU 05	**Acceder acciones agregadas**
Versión	1.0 (11-11-2023)
Actor	Usuario
Dependencias	● No aplica
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	Permite visualizar todas las acciones de la bolsa de valores que el usuario está siguiendo.
Secuencia normal de los eventos	1. El caso de uso inicia en el momento en que el usuario desea visualizar todas las acciones de la bolsa de valores que ha agregado. 2. El usuario se dirige al apartado “Acciones agregadas” y “Mostrar todas las acciones agregadas”. 3. El sistema muestra todas las acciones agregadas por el usuario de las que se tiene registro. (E-1) 4. Fin del caso de uso.
Postcondición	Despliegue de la información de las acciones del usuario en pantalla.
Excepciones	E-1: 3.1. El usuario no ha agregado ninguna acción. 3.1.1 Se muestra el mensaje de que no hay acciones agregadas. 3.1.2 Fin del caso de uso.
Comentarios	

CU 06	**Buscar acción agregada**
Versión	1.0 (11-11-2023)
Actor	Usuario
Dependencias	● No aplica
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	Permite visualizar una de las acciones de la bolsa de valores que el usuario está siguiendo.
Secuencia normal de los eventos	1. El caso de uso inicia en el momento en que el usuario desea visualizar una de las acciones de la bolsa de valores que ha agregado.

	2. El usuario se dirige al apartado “Acciones agregadas” y “Buscar por Id”. 3. El usuario ingresa el id de la acción. 4. El sistema valida si todos los campos obligatorios están llenos. (E-1) 5. Se habilita la opción de “Buscar”. 6. El usuario selecciona la opción “Buscar”. (E-2) 7. Se muestra la información de la acción ingresada. 8. Fin del caso de uso.
Postcondición	Despliegue de la información de la acción buscada por el usuario en pantalla.
Excepciones	E-1: 4.1 No todos los campos están llenos. 4.1.1 Los campos obligatorios faltantes se resaltan en rojo y el botón “Buscar” permanece deshabilitado. 4.1.2 Volver al paso 3. E-2: 6.1 No existe registro de la acción buscada en las acciones agregadas por el usuario. 6.1.1 Se informa al usuario. 6.1.2 Volver al paso 3.
Comentarios	

CU 07	**Eliminar Acción**
Versión	1.0 (11-11-2023)
Actor	Usuario
Dependencias	● No aplica.
Precondición	El usuario debe haber ejecutado anteriormente el caso de uso “Registrar Usuario”.
Descripción	Permite al usuario eliminar una de las acciones agregadas con anterioridad.
Secuencia normal de los eventos	1. El caso de uso inicia cuando el usuario desea eliminar una de las acciones que sigue. 2. El usuario se dirige a la opción de “Eliminar acción”. 3. El usuario ingresa la información de la acción a eliminar en los campos disponibles (Id acción). 4. El sistema valida si todos los campos obligatorios están llenos. (E-1) 5. Se habilita la opción “Eliminar”. 6. El usuario selecciona la opción “Eliminar”. (E-2)

	7. Fin del caso de uso.
Postcondición	La información de la acción eliminada por el usuario se borra de la Base de Datos.
Excepciones	E-1: 4.1 No todos los campos están llenos. 4.1.1 Los campos obligatorios faltantes se resaltan en rojo y el botón “Eliminar” permanece deshabilitado. 4.1.2 Volver al paso 3. E-2: 6.1 No existe registro de la acción a eliminar en las acciones agregadas por el usuario. 6.1.1 Se informa al usuario. 6.1.2 Volver al paso 3.
Comentarios	

1.2. Cualidades del sistema

1.2.1. Escenarios de calidad

Modificabilidad: Teniendo en cuenta los requisitos funcionales indicados se tiene que el sistema requiere la capacidad de cambiar la base de datos que almacena la información del mercado de valores con un esfuerzo igual al trabajo de dos personas durante un mes(2 personas-mes), Es necesario tener la capacidad de desarrollar nuevos indicadores de acciones, tales como gráficos diarios, mensuales y anuales, con un esfuerzo equivalente al trabajo de medio empleado durante un mes (0.5 personas-mes), todo lo anterior mencionado en tan solo una semana, estos cambios no indican que se debe desarrollar nuevas funcionalidades sino que por el contrario se integren con las funcionalidades existentes.

Cod. Escenario : EAC01	
Descripción: Surge la necesidad de modificar la base de datos para mejorar el rendimiento y la escalabilidad del sistema. La capa de abstracción de datos posibilita llevar a cabo la migración de la base de datos en una semana, con un esfuerzo equivalente al trabajo de dos personas a lo largo de un mes. En este proceso, se supervisa el tiempo real de ejecución de la migración, se detectan posibles errores y se evalúa la estabilidad del sistema tanto durante como después del cambio.	
Interesado: Desarrolladores y Administradores del sistema	**Atributo**: Modificabilidad
Validación del escenario	
Origen del estímulo	Mejorar el rendimiento
Estímulo	La necesidad de garantizar que las modificaciones sean eficientes y no afecten la funcionalidad actual del sistema.
Entorno	Mantenimiento del sistema
Artefacto	Bases de datos y capa de acceso a datos
Respuesta	La capa de abstracción de datos permite realizar la migración de la base de datos en una semana con un esfuerzo equivalente al trabajo de dos personas durante un mes, sin afectar la funcionalidad actual del sistema y mejorando el rendimiento y modificabilidad.
Medida de la respuesta	En tiempo real y lo más rápido posible

Cod. Escenario : EAC02	
Descripción: En la fase de desarrollo de nuevas funciones, se busca la incorporación de indicadores, como gráficos diarios, mensuales y anuales, para mejorar el análisis de acciones. La implementación de una API dedicada para generar y mostrar estos indicadores permite su integración en una semana, con un esfuerzo equivalente al trabajo de medio empleado durante un mes. Se controla el tiempo de desarrollo, la integración de los indicadores y se evalúa el rendimiento del sistema post incorporación.	
Interesado: Desarrolladores, Analistas y Administradores del sistema	**Atributo**: Modificabilidad
Validación del escenario	
Origen del estímulo	La necesidad de agregar nuevos indicadores, como gráficos diarios, mensuales y anuales, para mejorar el análisis de las acciones.
Estímulo	La necesidad de extender las capacidades de la aplicación sin comprometer su estabilidad.
Entorno	Desarrollar nuevas funcionalidades
Artefacto	API para generación y visualización de indicadores.
Respuesta	La implementación de la API permite la fácil incorporación de nuevos indicadores en una semana con un esfuerzo equivalente al trabajo de medio empleado durante un mes, sin comprometer la estabilidad ni el rendimiento del sistema.
Medida de la respuesta	En tiempo real y lo más rápido posible

Escalabilidad: Teniendo en cuenta los requisitos no funcionales indicados se tiene que el sistema debe ser capaz de soportar la interacción en un inicio con 100 usuarios simultáneos, y luego ir escalando a 900 y posteriormente 3000 usuarios, se plantean los siguientes escenarios de calidad:

Cod. Escenario : EAC03	
Descripción: Evaluar el sistema bajo diferentes escenarios de carga. Esto se haría por medio de una simulación en un inicio de 100 usuarios simultáneos realizando búsquedas y actualizaciones de precios para luego aumentar gradualmente la carga hasta 3,000 usuarios para representar el crecimiento a nivel nacional	
Interesado: Usuarios	**Atributo**: Escalabilidad
Validación del escenario	
Origen del estímulo	Conexiones de usuarios
Estímulo	Aumento de carga gradual
Entorno	Condiciones de aumento de carga
Artefacto	El sistema
Respuesta	El sistema debe mantener tiempos de respuesta aceptables en todos los niveles de carga, asegurando una experiencia de usuario fluida.
Medida de la respuesta	En tiempo real

Cod. Escenario : EAC04	
Descripción:evaluar la capacidad del sistema para redirigir el tráfico y recuperarse al introducir deliberadamente fallos en algunos servicios.	
Interesado: Usuarios	**Atributo**: Escalabilidad
Validación del escenario	
Origen del estímulo	Usuarios
Estímulo	Alto volumen de usuarios

Cod. Escenario : EAC04	
Descripción:evaluar la capacidad del sistema para redirigir el tráfico y recuperarse al introducir deliberadamente fallos en algunos servicios.	
Interesado: Usuarios	**Atributo**: Escalabilidad
Validación del escenario	
Entorno	Condiciones extremas de concurrencia
Artefacto	El sistema
Respuesta	El sistema debe ser tolerante a fallos, asegurando la continuidad del servicio incluso en presencia de eventos inesperados o la presencia de nuevos usuarios conectados constantemente.
Medida de la respuesta	Lo mas rápida posible

1.2.2. Escalabilidad

Las tácticas elegidas para respaldar el atributo de escalabilidad son:

- **Cost-Effectiveness:** Se refiere a la optimización de los recursos para lograr un equilibrio entre el rendimiento del sistema y los costos asociados. El objetivo es maximizar el rendimiento y la eficiencia del sistema sin llegar a tener costos innecesarios. Para esto, es importante realizar:
    - Un análisis detallado de los costos asociados con la infraestructura, el almacenamiento, el ancho de banda y otros recursos utilizados por la aplicación con el fin de saber a detalle lo que se necesita.
    - Evaluar el rendimiento en relación con los costos para identificar áreas donde se pueda mejorar la eficiencia, o en donde los recursos pueden llegar a sobrar debido al exceso de estos.
    - Escalamiento Proporcional: Ajustar la escala de recursos según la demanda del sistema de manera proporcional. Evitar la sobreasignación de recursos que no se utilizan completamente, lo cual podría resultar en costos innecesarios.
    - Optimización de Código: Realizar revisiones de código para identificar oportunidades de optimización que puedan reducir el uso de recursos, mejorando algoritmos y procesos.

De esta manera se logra optimizar los recursos utilizados y se evita el pago de costos que son innecesarios y que a largo plazo pueden afectar en la economía de la empresa. A la hora de escalar el sistema y tener en cuenta lo mencionado anteriormente , se pueden tomar decisiones informadas que inciden directamente en la calidad de software al momento de escalar y que sea para la empresa sea de manera costo-efectiva.

- **Load Balancing:** O balanceo de trabajo, busca distribuir la carga de trabajo de manera equitativa entre múltiples recursos, como servidores, nodos o instancias. El objetivo principal es prevenir la congestión en un único punto y asegurar que cada recurso contribuya de manera equitativa al procesamiento de solicitudes, para esto se debe tener en cuenta lo siguiente:
    - Distribución Equitativa: El balance de trabajo distribuye las solicitudes de manera equitativa entre varios servidores. Evita que un único recurso se sobrecargue mientras otros no se están utilizando completamente.
    - Algoritmos de Balanceo: Utiliza algoritmos específicos para determinar cómo se asignan las solicitudes a los recursos disponibles de manera que el sistema sea eficaz.
    - Escalabilidad Horizontal: Facilita el escalado horizontal al agregar más recursos al grupo de servidores. A medida que la carga aumenta, se pueden agregar nuevos servidores, y el balanceador de carga distribuirá automáticamente las solicitudes entre ellos.
    - Uso de balanceadores de carga: Si un servidor falla, el balanceador de carga puede redirigir automáticamente las solicitudes a otros servidores, minimizando el impacto en el servicio.
    - Monitoreo de Recursos: Usar sistemas de monitorización para revisar la carga en cada recurso y de esta forma tomar decisiones dinámicas basadas en el rendimiento actual de los servidores, redistribuyendo las solicitudes para optimizar el rendimiento.
    - Seguridad: Puede contribuir a la seguridad al actuar como un punto de entrada único, filtrando y distribuyendo el tráfico malicioso antes de llegar a los servidores.

  De esta manera se busca que todos los componentes trabajen por igual sin que haya un sobre esfuerzo o tráfico pesado sobre un módulo específico.

Estas dos estrategias se complementan de gran manera ya que una nos permite no sobrepasarse con lo recursos que se tienen evitando gastos innecesarios y la otra nos ayuda a como usar estos recursos apropiadamente, todos con una misma carga de trabajo para que el software trabaje de manera equilibrada. Y si se da el caso aumentar nuevos recursos de manera efectiva y sin costos adicionales.

La escalabilidad de la arquitectura cliente-servidor se atribuye a su capacidad para distribuir y asignar responsabilidades entre el cliente y el servidor de manera efectiva. La clara división de funciones posibilita la expansión horizontal mediante la incorporación de

servidores para gestionar eficientemente el aumento de la carga. Los servidores especializados en tareas específicas, como procesamiento de datos o administración de bases de datos, simplifican la distribución de la carga y la implementación de estrategias de equilibrio de carga. Además, esta estructura fomenta el desarrollo modular, posibilitando actualizaciones y mejoras sin afectar la estabilidad general del sistema, lo que resulta en una escalabilidad más eficiente a medida que las exigencias del sistema evolucionan. Esta arquitectura es una excelente elección si se busca que nuestro sistema sea escalable.

1.2.3. Modificabilidad

Las tácticas elegidas para respaldar el atributo de modificabilidad son:

- **Split a Responsibility:** Dada la necesidad de que la aplicación sea flexible frente a los cambios en la base de datos y la introducción de nuevos indicadores, la asignación de responsabilidades específicas, como la gestión de la base de datos y la generación de indicadores, puede reducir el costo asociado con la modificación individual de cada una de estas tareas. Esto proporciona una mayor modularidad y facilita la adaptación a cambios específicos sin afectar otras partes del sistema.
    - Facilita la identificación y modificación de componentes específicos sin afectar al conjunto completo del sistema.
    - Aumenta la transparencia y facilita la comprensión del código mediante la descomposición de tareas complejas en tareas más manejables.
    - Facilita los cambios locales y evita hacer cambios extensos en el sistema
    - Estructura el sistema en capas, asignando a cada capa una responsabilidad definida. Esto simplifica la distribución de tareas a nivel de cada capa.
    - "Dividir una responsabilidad" podría indicar la división de la gestión de la base de datos y la generación de indicadores en componentes independientes, lo cual aplicado a la gestión de bases de datos y generador de indicadores sería óptimo ya que cada módulo tendría una responsabilidad clara para adaptarse a cambios específicos en esas áreas sin afectar el conjunto del sistema.
- **Maintain Semantic Coherence:** Es una estrategia de diseño que se enfoca en preservar la consistencia y el significado lógico en el sistema, especialmente cuando se introducen cambios para nuestro caso mantener la coherencia semántica es esencial para comprender y modificar el sistema de forma eficaz. Al reunir funciones relacionadas, como la generación de indicadores y la gestión de la base de datos, se refuerza la cohesión, lo que simplifica la comprensión y modificación de módulos específicos sin influir en otros. Esto conlleva a una mayor facilidad en las labores de mantenimiento y ajuste.
    - Asegurar que las modificaciones no introduzcan inconsistencias en el significado lógico de las operaciones y funcionalidades del sistema.
    - Minimiza los cambios que puedan tener efectos no deseados en otras partes del sistema.
    - Implementa pruebas exhaustivas para verificar la coherencia semántica antes y después de realizar modificaciones. Las pruebas deben cubrir no

solo las funcionalidades existentes, sino también cualquier nueva funcionalidad introducida.
    - Los desarrolladores y los usuarios pueden confiar en que las modificaciones realizadas en el sistema no alterarán su comportamiento de una manera que cause confusión o errores.
    - La coherencia semántica facilita el mantenimiento del sistema, ya que los desarrolladores pueden entender y prever cómo las modificaciones afectarán la funcionalidad existente.

Estas dos estrategias se complementan de gran manera ya que estas estrategias están en sintonía con la necesidad de una aplicación que pueda ajustarse sin complicaciones a modificaciones en la base de datos y la incorporación de nuevos indicadores. Proporcionan una estructura modular y consistente que simplifica la comprensión y la modificación del sistema.

La arquitectura hexagonal, también conocida como puertos y adaptadores, ofrece beneficios notables en términos de modificabilidad y escalabilidad. La separación clara entre la lógica de la aplicación y las interfaces externas promueve un desacoplamiento fuerte, permitiendo modificaciones en la lógica interna sin afectar las interfaces externas. Esto facilita la adaptación a cambios en los requisitos y la evolución de la aplicación. La flexibilidad en la lógica de la aplicación, independiente de la implementación de interfaces externas, simplifica la incorporación de nuevas funcionalidades. Además, la estructura modular permite la escalabilidad horizontal al escalar componentes específicos sin afectar otras partes del sistema. La adaptabilidad a cambios de requisitos se ve facilitada por la arquitectura hexagonal, que permite agregar nuevos servicios o adaptadores para abordar cambios en el entorno. En resumen, esta arquitectura proporciona un entorno desacoplado y flexible que favorece la modificabilidad y escalabilidad al adaptarse eficientemente a cambios y manejar cargas variables de manera modular. Todo esto permite la incorporación de funcionalidades y la adaptación a los cambios sin afectar a las demás partes del sistema, lo que permite que la aplicación evolucione con el tiempo.

2. Arquitectura del sistema

2.1. Enfoque ADD

Considerando que los atributos de calidad con los que cuenta el sistema son la escalabilidad y modificabilidad, se da la necesidad de priorizar uno de estos para la toma de decisión de una arquitectura la cual más favorezca el desempeño general del sistema. Sin embargo, esto puede resultar algo complejo pues ambos cuentan con un alto grado de importancia en la especificación de las cualidades del sistema.

Es debido a esto que se tiene que entrar en el análisis de los escenarios en el peor de los casos de falla para cada uno de estos atributos, para de esta manera determinar cual conlleva a una mayor pérdida en cuanto a valor para el sistema.

En primer lugar se tiene el atributo de modificabilidad, el cual nos permite minimizar el esfuerzo de personas-mes para la adición de nuevas funcionalidades al sistema, donde en caso de no contar con este, significaría un mayor gasto de recursos monetarios, de esfuerzo y de tiempo. Recursos los cuales a pesar de generar una pérdida, podrían soportarse, puesto que el servicio que se está prestando tiene la manera de pagarse por sí mismo, sólo se llegaría a reducir las ganancias que genera.

Por otro lado, el atributo de escalabilidad nos permite una mejor gestión y alta tolerancia a fallos ante una alta concurrencia de usuarios. Por lo que en caso de verse perjudicada significaría la falla del sistema y pérdida de confiabilidad de los usuarios, lo cual es difícilmente recuperable.

De esta manera, podemos observar cómo a pesar de la importancia de la modificabilidad, sus consecuencias se podrían solventar más fácilmente ante la falta de una correcta escalabilidad del sistema. Es por esta razón que el driver del sistema viene siendo dado por la escalabilidad.

En cuanto al patrón arquitectónico a elegir, debemos considerar las tácticas especificadas con anterioridad para lograr el driver elegido, el cual es la escalabilidad. Entre estas, tenemos el balanceo de cargas, lo que significa que se necesita un patrón arquitectónico el cual nos permite la división de la carga de trabajo de tal manera que se prevenga la congestión en un único componente, además de permitir una escalabilidad horizontal de forma que el sistema sea capaz de manejar un aumento en la demanda. Uno de los mejores patrones arquitectónicos que nos brindan estas características es el de publicador/suscriptor.

En este patrón, los componentes de productores y consumidores están desacoplados, lo que conlleva a que estos puedan escalar y evolucionar de manera independiente, permitiendo así, agregar más instancias de productores y consumidores según sea necesario sin afectar significativamente el rendimiento del sistema, ni la lógica interna de los demás componentes. Además, permite un procesamiento en paralelo eficiente donde múltiples consumidores pueden procesar eventos simultáneamente, lo que mejora la capacidad de respuesta y el rendimiento general del sistema.

Lo anterior resulta bastante beneficioso para el software de seguimiento del mercado de valores, donde con la evolución de este, se irá expandiendo a nivel regional y por lo tanto utilizando progresivamente por un número mayor de usuarios. Esto debido a que, ante un aumento en la demanda de clientes, es posible ir aumentando la cantidad de consumidores, los cuales simplemente se unen al sistema. A su vez, debido a que los nuevos usuarios pueden encontrarse utilizando el sistema de manera simultánea, con la arquitectura publicador/suscriptor también se logra una resistencia a picos en la carga del sistema, pues los consumidores pueden procesar eventos a su propio ritmo, y la arquitectura distribuida facilita la absorción de estas cargas variables.

Así, la arquitectura publicador/suscriptor facilita la escalabilidad al permitir la adición de nuevos componentes de manera independiente y distribuida, brinda una mejor capacidad para manejar cargas variables y se adapta rápidamente a cambios

inesperados en el sistema, siendo por lo tanto, el patrón más indicado para la arquitectura del sistema de seguimiento del mercado de valores.

2.2. Perspectivas

Dados los requisitos de la aplicación se tiene la posibilidad de ir escalando poco a poco las vistas para alcanzar una mejor descripción del sistema, desde una vista general a una más específica, como se verá en los respectivos diagramas en los siguientes puntos.

La primera perspectiva viene dada desde la vista de **componentes y conectores**. Al tener el sistema la necesidad de generar notificaciones de manera asíncrona de un gran número de acciones y sus impredecibles variaciones en precio, se opta en esta vista por una arquitectura Publicador - Suscriptor. Partiendo de lo explicado anteriormente, el Suscriptor serán todos los interesados en el uso de la aplicación, por lo que puede haber un gran número de usuarios simultáneos. En nuestro contexto de desarrollo, el aplicativo SuAgenteFinanciero cumple el rol de suscriptor. Dentro de este se hace la interacción humano-máquina y recibe las notificaciones específicas en los cambios de precio que superen el umbral establecido por el usuario.

En lo que respecta al publicador, el cual es el apartado de énfasis en el desarrollo de esta aplicación, se tendrá la lógica necesaria para la observación de los cambios en los precios de las acciones en las bases de datos externas a las que se tiene acceso y el respectivo armado de los mensajes a enviar a los suscriptores interesados en las acciones al superar los umbrales establecidos.

Este juego de Publicador suscriptor se realizará con ayuda de RabbitMQ con su protocolo AMQP. Es gracias a esto que se permite la notificación asíncrona de los cambios de precio en las acciones. Brindando de esta manera, la seguridad de que los diferentes usuarios (por medio de los suscriptores) serán correctamente notificados y, de paso, de una forma mas optima, pues no tendrán que estar en todo momento haciendo peticiones a un servidor para revisar si algún umbral se superó, únicamente se opta por esperar a que llegue un mensaje a la cola de mensajes.

Con este especial énfasis en el publicador, mismo apartado en el que se centra el desarrollo de este proyecto, y que se estudiará más en detalle de acá en adelante, pasamos a la siguiente perspectiva, la de desarrollo, basándonos en una vista de **módulos**.

Desde el punto de vista de módulos debe considerarse el patrón de arquitectura limpia (también conocida como puertos y adaptadores, domain-driven, hexagonal). La razón para esto es priorizar la seguridad de las peticiones hechas al servidor sobre el rendimiento, dado que un fallo en la seguridad de la información podría resultar en pérdidas financieras para los usuarios de la aplicación, dada la naturaleza de la misma.

La estructura del publicador sigue, entonces, esta arquitectura hexagonal:

1. **Puertos (Ports):**

   Responsabilidad: Definen las interfaces de interacción con el exterior. En este caso, la interfaz entre el cliente y el resto de la aplicación.

   Funciones Principales:

   - Procesa y enruta las solicitudes del cliente.
   - Define interfaces para validar la entrada del cliente y realizar operaciones específicas.

2. **Adaptadores (Adapters):**

   Responsabilidad: Implementan las interfaces definidas en los puertos y conectan la aplicación con el exterior.

   Funciones Principales:

   - Adaptan las solicitudes HTTP para ser procesadas internamente.
   - Convierten las respuestas internas a un formato adecuado para ser enviadas al cliente en JSON.

3. **Lógica de Aplicación (Application Logic):**

   Responsabilidad: Contiene la lógica de negocio central de la aplicación de la bolsa de valores.

   Funciones Principales:

   - Coordina las operaciones necesarias en respuesta a las solicitudes del cliente.
   - Aplica la lógica de negocio y las reglas del dominio de la aplicación.
   - Interactúa con la capa de acceso a datos.

4. **Adaptadores de Datos (Data Adapters):**

   Responsabilidad: Implementan interfaces para interactuar con la base de datos u otros sistemas de almacenamiento.

   Funciones Principales:

   - Realizan operaciones de lectura y escritura en la base de datos.
   - Abstraen la complejidad de las consultas SQL u otras operaciones de almacenamiento.

- Proporcionan métodos para acceder a los datos de manera eficiente y segura.

5. **Dominio (Domain):**

   Responsabilidad: Contiene la lógica de negocio principal y representa el núcleo conceptual de la aplicación de la bolsa de valores.

   Funciones Principales:

   - Define los objetos de dominio y las reglas de negocio.
   - Encapsula la lógica que no pertenece ni a la capa de aplicación ni a la capa de adaptadores de datos.
   - Contiene clases y métodos que representan entidades de negocio como acciones y usuarios.

Finalmente, el despliegue del sistema se hará en 4 diferentes componentes: publicador, suscriptor, base de datos de usuarios y base de datos externa de las acciones en el mercado. Cada uno de estos se encontrará en nodos diferentes. El suscriptor desde la conexión del usuario, el publicador principal en un servidor de aplicaciones físico, la base de datos de los usuarios igual, y la base de datos de las acciones será externa, con un proveedor de ese servicio.

2.3. Modelo C4

Cada uno de los diagramas de C4 se encuentran en los anexos con sus respectivos nombres.

2.3.1. Contexto

Anexo [D_Context](D_Context.pdf)

2.3.2. Contenedores

Anexo [D_Containers](D_Containers.pdf)

2.3.3. Componentes

Anexo [D_Components](D_Components.pdf)

2.3.4. Codigo

Anexo [D_Modules_Server](D_Modules_Server.pdf)

Anexo [D_Modules_Commons](D_Modules_Commons.pdf)

2.4. Diagramas UML

Cada uno de los diagramas UML se encuentran en los anexos con sus respectivos nombres.

2.4.1. Diagrama de componentes y conectores

Anexo [D_C&C](D_C&C.pdf)

2.4.2. Diagrama de Módulos

Anexo [D_Modules_Server](D_Modules_Server.pdf)

Anexo [D_Modules_Commons](D_Modules_Commons.pdf)

2.4.3. Diagrama de Secuencia

Anexo [D_Secuencial](D_Secuencial.pdf)

3. Esqueleto del Sistema usando Spring Boot

3.1. Aplicación

En anexos.

3.2. Prueba de servicios

En anexos.

Referencias

- Nanda, S. P., & Reza, H. (n.d.). Scalability Tactics in Data-Intensive Systems.
- Bachmann, F., Bass, L., & Nord, R. (September 2007). Modifiability Tactics. *Software Engineering Institute*.

Anexos

- Implementación del Sistema en NetBeans usando Spring Boot.
- Prueba de servicios: Peticiones Postman.
- Prueba de servicios: HTTP Swagger: http://localhost:8080/doc/swagger-ui/index.html
- [D_Context](D_Context.pdf)
- [D_Containers](D_Containers.pdf)
- [D_Components](D_Components.pdf)
- [D_C&C](D_C&C.pdf)

- [D_Modules_Server](D_Modules_Server.pdf)
- [D_Modules_Commons](D_Modules_Commons.pdf)
- [D_Secuencial](D_Secuencial.pdf)