PROYECTO #1

MESSAGE QUEUE

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Algoritmos y Estructuras de Datos I (CE 1103)
I Semestre 2024

OBJETIVOS

GENERAL

- Implementar un sistema de mensajería para comunicar asincrónicamente aplicaciones distribuidas

ESPECÍFICOS

- Implementar las diferentes estructuras de datos lineales (listas, pilas y colas).
- Desarrollar algoritmos para dar solución a un problema.
- Fomentar la creatividad mediante el análisis y diseño de algoritmos
- Utilizar diagramas de clases UML para modelar una solución a un problema.
- Aplicar patrones de diseño en la elaboración de una solución a un problema.

REQUERIMIENTOS

Un sistema distribuido se puede explicar en términos sencillos, como un grupo de programas que se ejecutan en diferentes máquinas (físicas o virtuales) y se comunican mediante protocolos de red bien conocidos. Por ejemplo, Netflix es un ecosistema de aplicaciones con responsabilidades claras que interactúan entre sí para llevar a cabo los requerimientos de los usuarios. Netflix tiene un servicio que se ejecuta en un determinado grupo de servidores para realizar el streaming de los bits del video, otro servicio que se encarga de las recomendaciones, otro servicio lleva el control de las estadísticas de uso y así sucesivamente. Cada servicio se comunica con otros mediante la red.

Hay muchas formas de lograr que las aplicaciones “hablen” entre sí. La comunicación punto a punto es la más sencilla. Este tipo de comunicación permite que la aplicación A, invoque código dentro de la aplicación B (A y B en máquinas aparte) a través de un endpoint. Aunque este es un enfoque muy sencillo, no escala bien.

Otra técnica muy comúnmente usada es utilizar un *middleware*, es decir, un sistema que permite comunicar sistemas independientes, sin que estos sepan nada uno del otro. El *middleware* actúa como ese intermediario para comunicar las aplicaciones.

En este proyecto, usted construirá un *middleware* orientado a colas de mensajes. Las colas de mensajes permiten a un sistema, depositar un mensaje que solicita que cierta acción se lleve a cabo en algún momento por alguna entidad desconocida. Las colas de mensajes se entienden como comunicación asincrónica, es decir, la petición y la respuesta pueden ocurrir en momento lejanos en el tiempo.

Este proyecto tiene varias partes. La **primera parte** es un programa que deberá crear llamado MQBroker. Es un proyecto consola en .NET que se encarga de gestionar las colas y transmitir los mensajes entre las colas. MQBroker utiliza una topología publicador/subscriptor. MQBroker utiliza un socket servidor para escuchar las peticiones de las aplicaciones que envían/consumen mensajes. A continuación se listan los requerimientos para este programa:

ID	DESCRIPCIÓN	PUNTOS
001	Al iniciar, se activa un socket para escuchar peticiones entrantes de los clientes (aplicaciones que envían/reciben mensajes). El formato de los mensajes será definido por los estudiantes. Debe tener un formato consistente para todas las peticiones.	10
002	Escucha una petición llamada *Subscribe*. La petición incluye: • AppID en formato GUID generador por la aplicación que envía la petición • Tema al que se quiere subscribir Al recibir esta petición, MQBroker verifica si dicho AppID está o no subscrito al tema. Si está subscrito, retorna un mensaje que lo indique. Si no, crea una cola exclusiva para dicho tema y dicho AppID.	10
003	Escucha una petición llamada *Unsubscribe*. La petición incluye: • AppID en formato GUID generador por la aplicación que envía la petición • Tema al que se quiere subscribir Al recibir esta petición, MQBroker verifica si dicho AppID está o no subscrito al tema. Si no está subscrito, retorna un mensaje que lo indique. Si no, elimina la cola para dicho tema y dicho AppId.	10
004	La petición *Publish* incluye*:* • AppID en formato GUID de la aplicación que publica el mensaje • Tema al que se desea publicar en formato string. Por ejemplo: Tema/TipoDeCambio/Dolares • Contenido de la publicación, es decir el cuerpo del mensaje como tal Cuando el broker recibe una petición publish, verifica que el tema exista. Si no existe, ignora la publicación. Si el tema existe, por cada uno de los subscriptores para dicho tema, coloca un mensaje en su cola respectiva	10
005	La petición *Receive* permite a una aplicación específica, obtener un mensaje de la cola. Es decir, los mensajes colocados en la cola por el broker, se	10

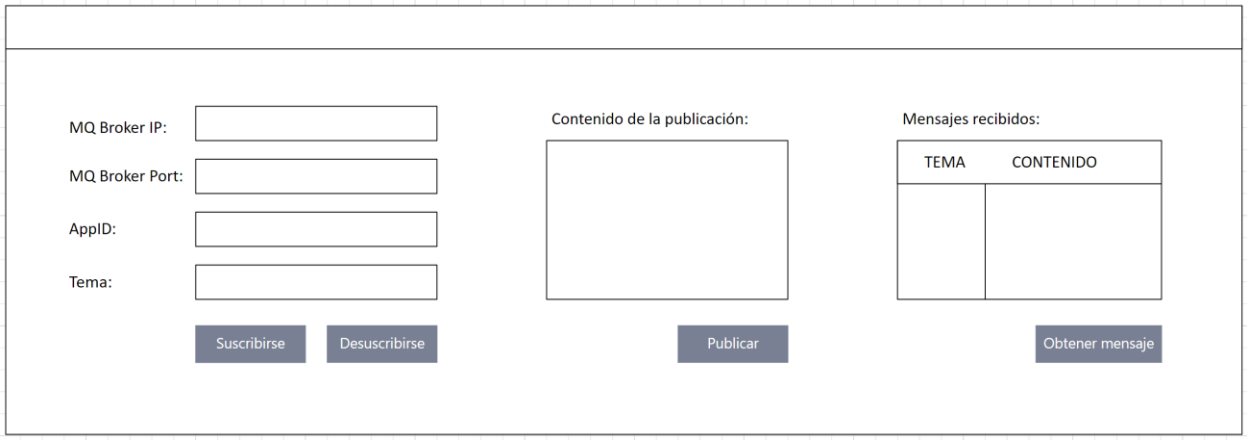
	quedan en la cola hasta que la aplicación correspondiente solicite obtener dicho mensaje. La petición *receive* incluye: • AppID en formato GUID de la aplicación que desea obtener el mensaje y que previamente se suscripción • Tema en formato string del que desea obtener un mensaje Al recibir la petición, el broker verifica que el AppID esté subscrito a dicho mensaje, si no lo está retorna un error. De lo contrario, verifica si hay algún mensaje en la cola de dicho AppID para ese tema. Si no hay mensaje, retorna un error. Si hay un mensaje, lo saca de la cola (FIFO) y lo retorna a la aplicación a través del socket	

La **segunda parte** es una biblioteca (class library) en .NET que permite a los clientes, interactuar con el broker. Contiene la clase facade MQClient. Dicha clase contiene al menos los siguientes métodos públicos:

ID	DESCRIPCIÓN	PUNTOS
001	**MQClient(string ip, int port, Guid AppID)** Constructor que crea un nuevo MQClient. Recibe el IP y el puerto donde está escuchando el MQBroker. Además recibe el AppID (generado por la aplicación que utiliza la bibilioteca MQClient). Este método puede realizar cualquier inicialización requerida por la biblioteca.	10
002	**bool Subscribe(Topic topic)** Envia la petición Subscribe al MQBroker mediante un socket de tipo cliente. Nótese que el argumento es un objeto tipo Topic. Este es un objeto definido por ud que encapsula un string que contiene el tema. La idea es que al utilizar un objeto de nivel más alto, se pueden realizar validaciones y otras modificaciones relacionadas con temas. Retorna true o false indicando si la subscripción se realizó o no. Debe lanzar excepciones relevantes según aplique.	5
003	**bool Unsubscribe(Topic topic)** Envía la petición Unsubscribe al MQBroker mediante un socket de tipo cliente. Retorna true o false indicando si se pudo o no de suscribir del tema. Debe lanzar excepciones relevantes según aplique.	5
004	**bool Publish(Message message, Topic topic)** Envía la petición Publish al MQBroker mediante un socket de tipo cliente. El objeto Message encapsula el mensaje o contenido de la publicación e incluye cualquier funcionalidad necesaria para serializar/deserializar un mensaje como se recibió del socket. Retorna true o false indicando si el mensaje se pudo colocar o no. Debe lanzar excepciones relevantes según aplique.	5

005	**Message Receive(Topic topic)** Envía la petición Receive al MQBroker mediante un socket de tipo cliente. Retorna un objeto Message con el contenido del mensaje. Debe lanzar excepciones relevantes según aplique.	5

La **tercera parte** es un programa con interfaz gráfica en .NET Core que permitirá probar las funcionalidades del sistema MQBroker. **Tendrá un valor de 20 puntos**. La interfaz gráfica puede ser similar a la siguiente (con las funcionalidades mínimas):



DOCUMENTACIÓN

- La documentación deberá tener las partes estándar:
  - Portada
  - Introducción
  - Tabla de contenidos (con los títulos debidamente numerados)
  - Breve descripción del problema
  - Descripción de la solución: por cada uno de los requerimientos, se deberá explicar cómo se implementó, alternativas consideradas, limitaciones, problemas encontrados y cualquier otro aspecto relevante.
  - Diseño general: diagrama de clases UML con las clases relevantes que muestren el diseño orientado a objetos y los patrones de diseño aplicados

ASPECTOS OPERATIVOS

- El proyecto tiene un valor de 25% de la nota del curso
- El trabajo se realizará en grupos de **2 personas**
- El uso de Git y Github es obligatorio
- La fecha de entrega será según lo especificado en el TEC Digital. Se entrega en el TEC digital, un archivo PDF con la documentación. Los estudiantes pueden seguir trabajando en el código hasta 15 minutos antes de la cita revisión oficial.

- Los proyectos que no cumplan con los siguientes requisitos no serán revisados:
    - Toda la solución debe estar integrada
    - La interfaz de usuario debe estar implementada e integrada
- El código tendrá un valor total de 80%, la documentación 10% y la defensa 10%. De estas notas se calculará la *Nota Final del Proyecto*.
- **Si no se entrega documentación en formato PDF, automáticamente se obtiene una nota de 0.**
- Si no se utiliza un manejador de código se obtiene una nota de 0.
- Si la documentación no se entrega en la fecha indicada se obtiene una nota de 0.
- El código debe desarrollarse en C#, si no, se obtendrá una nota de 0.
- La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.
- Cada estudiante tendrá 15 minutos para exponer su trabajo al profesor y defenderlo, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo que se recomienda tener todo listo antes de entrar a la defensa.
- Cada grupo es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.
- Durante la revisión únicamente podrán participar los miembros del grupo, asistentes, otros profesores y el coordinador del área.
- Cualquier estructura de datos requerida, deberá ser implementada por los estudiantes. No se permite el uso de estructuras de datos como listas, colas, árboles o cualquier otra provista por .NET