Trabajo Práctico

Fecha de Entrega: 28 de junio de 2022

Introducción

AlGlobo.com es un nuevo sitio de venta de pasajes online. Gracias al fin de la pandemia ha empezado a crecer fuertemente y necesitan escalar su sistema de procesamiento de pagos.

Para ello desean reemplazar una implementación monolítica actual con un microservicio en Rust que se encarge especificamente de este proceso.

Objetivo

Deberán implementar un conjunto de aplicaciones en Rust que modele el sistema de procesamiento de pagos de AlGlobo.com.

Se debe implementar un proceso para cada una de las entidades intervinientes y estas se comunicarán entre sí por sockets.

Se debe poder simular la salida de servicio de cualquiera de los procesos y réplicas de forma aleatoria o voluntaria, mostrando que el sistema en su conjunto sigue funcionando.

Requerimientos

El proceso de pagos se compone de la siguiente manera:

- Intervienen 4 entidades: AlGlobo.com, el banco, la aerolínea y el hotel.
- Existe una cola de pagos a procesar que se lee desde un archivo.
- AlGlobo.com debe coordinar el pago informando el monto a cobrar a cada entidad de forma concurrente.
- Cada entidad puede aleatoriamente procesar correctamente el cobro o no.
- Si alguna falla, se debe mantener la transaccionalidad y por lo tanto revertir o cancelar apropiadamente.
- Las fallas se guardan en un archivo de fallas para su posterior procesamiento manual. Debe implementarse una utilidad que permita reintentar manualmente cada pedido

fallado.
- El sistema de AlGlobo.com es de misión crítica y por lo tanto debe mantener varias réplicas en línea listas para continuar el proceso, aunque solo una de ellas se encuentra activa al mismo tiempo. Para ello utiliza un algoritmo de elección de líder y mantiene sincronizado entre las réplicas la información del la transacción actual.
- La aplicación de AlGlobo.com esta basada internamente en el modelo de Actores, utilizando el framework Actix.
- Todas las aplicaciones deben escribir un archivo de log con las operaciones que se realizan y sus resultados.
- El sistema mantendrá estadísticas operacionales. Para ello debe calcular el tiempo medio que toma un pago en procesarse desde que ingresa el pedido hasta que es finalmente procesado por todas las entidades.

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución de los ejercicios:

- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- No se permite utilizar **crates** externos, salvo los explícitamente mencionados.
- El código fuente debe compilarse en la última versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar **warnings** del compilador, ni del linter **clippy**.
- Las funciones y los tipos de datos (**struct**) deben estar documentadas siguiendo el estándar de **cargo doc**.
- El código debe formatearse utilizando **cargo fmt**.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

Entrega

La resolución del presente proyecto es en grupos de tres integrantes.

La entrega del proyecto comprende lo siguiente:

- Informe, se deberá presentar en forma digital (PDF) enviado por correo electrónico a la dirección: pdeymon@fi.uba.ar

- El código fuente de la aplicación, que se entregará únicamente por e-mail. El código fuente debe estar estructurado en un proyecto de cargo, y se debe omitir el directorio target/ en la entrega. El informe a entregar debe contener los siguientes items:
  - Una explicación del diseño y de las decisiones tomadas para la implementación de la solución.
  - Detalle de resolución de la lista de tareas anterior.
  - Diagrama que refleje los threads, el flujo de comunicación entre ellos y los datos que intercambian.
  - Diagramas de entidades realizados (structs y demás).

Criterios de evaluación

Presentación, principios teóricos y defensa de bugs potenciales

Los alumnos presentarán el código de su solución en vivo en una reunión sincrónica, con foco en el uso de las diferentes herramientas de concurrencia. Deberán poder explicar desde los conceptos teóricos vistos en clase cómo se comportará potencialmente su solución ante problemas de concurrencia (por ejemplo ausencia de deadlocks).

En caso de que la solución no se comportara de forma esperada, deberán poder explicar las causas y sus posibles rectificaciones.

Casos de prueba en vivo

Durante la presentación se someterá a la aplicación a diferentes casos de prueba que validen la correcta aplicación de las herramientas de concurrencia.

Informe

El informe debe estar estructurado profesionalmente y debe poder dar cuenta de las decisiones tomadas para implementar la solución.

Se debe detallar en un diagrama, las entidades desarrolladas, las herramientas de concurrencia empleadas. Así como también los threads y formas de comunicación entre ellos. Se debe poder entender qué mensajes datos entre ellos y de qué forma.

Organización del código

El código debe organizarse respetando los criterios de buen diseño y en particular aprovechando las herramientas recomendadas por Rust (i.e. no utilizar unsafe)

Tests automatizados

La presencia de tests automatizados que prueben diferentes casos, en especial sobre el uso de las herramientas de concurrencia es un plus.

Presentación en término

El trabajo deberá entregarse para la fecha estipulada. La presentación fuera de término sin coordinación con antelación con el profesor influye negativamente en la nota final.

Participación individual

Si bien el trabajo es grupal, la nota es individual y la participación del alumno durante la presentación influye en su nota final.