Facultad de Ingeniería

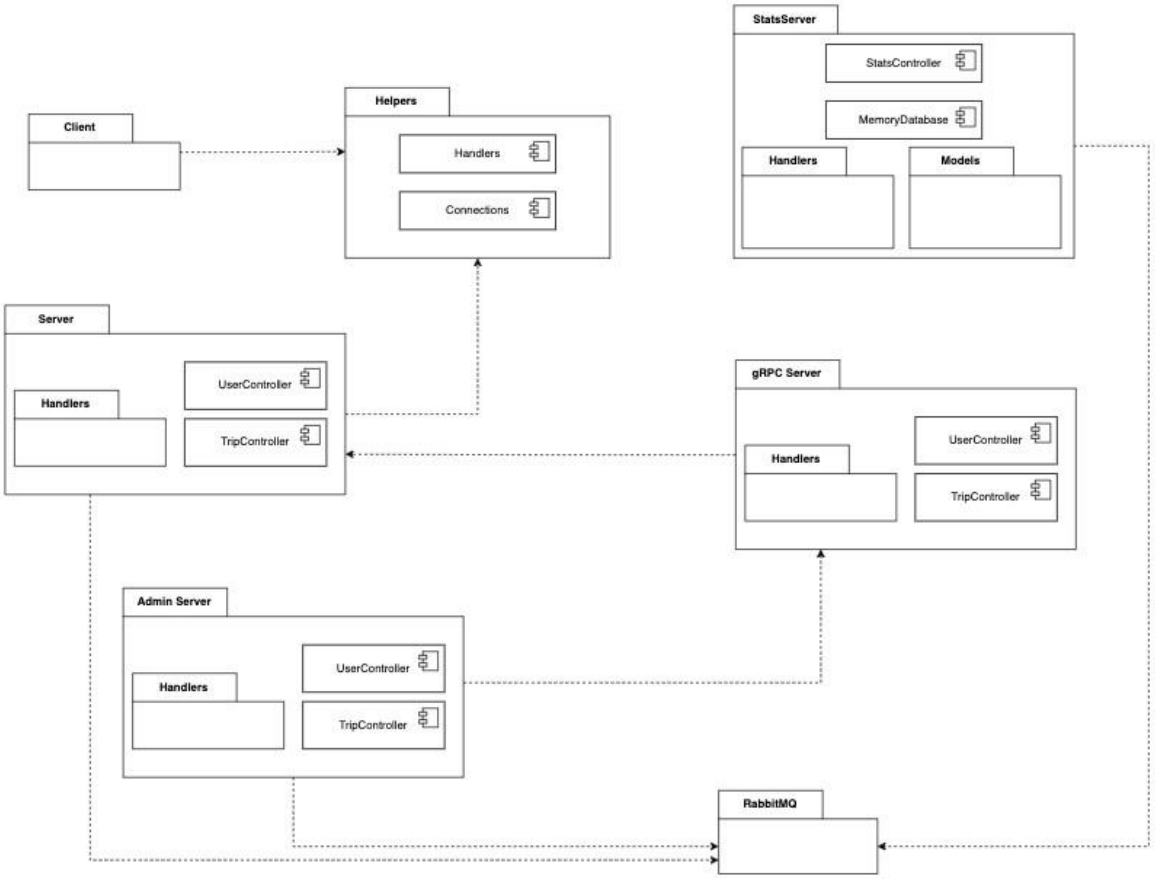
3er Obligatorio Programación de Redes

17 de Junio de 2024

Índice.

Diagrama de Componentes:

Se adjunta un diagrama detallando la composición de los componentes del proyecto. En esta entrega se agregan los módulos: AdminServer, GRPC Server y StatsServer.



Tecnologías

Se pedía que se utilizaran tres tecnologías: gRPC, RabbitMQ y REST API, y que se integraran las tres en al menos una aplicación.

gRPC

gRPC es un framework de llamada a procedimientos remotos (RPC) de alto rendimiento desarrollado por Google. Utiliza HTTP/2 para el transporte y Protocol Buffers (protobufs) para la serialización de datos, permitiendo comunicación eficiente y rápida entre clientes y servidores. Es especialmente útil para servicios distribuidos y microservicios.

Características clave:

- Comunicación de alto rendimiento y baja latencia.
- Soporte multi-idioma (C++, Java, Python, Go, Ruby, etc.).
- Comunicación bidireccional en tiempo real mediante streaming.
- Generación automática de código a partir de archivos .proto.

REST API

REST (Representational State Transfer) es un estilo de arquitectura para servicios web que utiliza HTTP y se basa en recursos identificados por URLs, manipulados mediante operaciones HTTP estándar (GET, POST, PUT, DELETE). Es ampliamente utilizado por su simplicidad y compatibilidad.

Características clave:

- Simplicidad y compatibilidad con la mayoría de lenguajes y entornos.
- Escalabilidad en aplicaciones distribuidas.
- Stateless: cada solicitud contiene toda la información necesaria.
- Flexibilidad para representar recursos en formatos como JSON, XML, HTML.

RabbitMQ

RabbitMQ es un broker de mensajes de código abierto que implementa el protocolo AMQP. Actúa como intermediario para enviar mensajes entre sistemas o componentes de una aplicación, manejando patrones de mensajería como colas y pub/sub, ideal para comunicación asíncrona y confiable en sistemas distribuidos.

Características clave:

- Facilita la comunicación asíncrona entre componentes.
- Alta disponibilidad y confiabilidad con clustering y replicación de colas.
- Escalabilidad horizontal mediante clustering.
- Soporte multi-protocolo (AMQP, MQTT, STOMP, HTTP).

En nuestro proyecto, hemos decidido emplear una combinación de tecnologías para optimizar la comunicación entre nuestros servidores. Utilizamos gRPC para la comunicación de alto rendimiento y baja latencia entre el Servidor de Administrador y el Servidor TCP. Empleamos una REST API para obtener datos relacionados con las estadísticas desde el Servidor de Estadísticas, aprovechando su simplicidad y compatibilidad. Además, implementamos RabbitMQ para manejar la comunicación asíncrona, específicamente entre el Servidor de Estadísticas y el Servidor TCP, así como entre el Servidor de Administrador y el Servidor TCP. Esta combinación nos permite construir un sistema robusto, escalable y eficiente.

Cambios

Comunicación y Gestión de Usuarios y Viajes

- **Formato de las Colas**: Utilizamos colas para enviar y recibir información de usuarios y viajes a través de RabbitMQ, asegurando una transmisión eficiente y asíncrona.
- **Locks**: Implementamos mecanismos de bloqueo para garantizar la consistencia de los datos durante las operaciones concurrentes.Tuvimos que aplicar locks principalmente en lugares donde persistían objetos, ya que en esas clases se implementan los funciones CRUD sobre los elementos. Un ejemplo de esto es cuando se agregan elementos con un id, ya que el contador del mismo es global y al agregar se incrementa(Trip y Reporte poseen Id), el uso del lock nos ayuda a que solo un hilo acceda a esta función.
- **Referencia de Usuario en el Handler**:Tuvimos un problema con el IHandler debido a que varias funciones se cambiaron a async Task. Esto hizo imposible pasar el usuario logueado usando ref. Para resolver esto, tuvimos que encapsular el usuario logueado en un "contenedor" para mantener la referencia correcta del objeto.

Funcionalidades del Servidor de Administración

- **Funciones Adicionales para el Administrador**: Añadimos funciones específicas para el administrador del servidor, permitiendo un control más detallado y la gestión de usuarios y viajes. Hicimos este cambio porque no se podía acceder correctamente a las funcionalidades implementadas para el TcpClient. El acceso a las funcionalidades proporcionadas por IHandler y el servidor estaba demasiado restringido, ya que siempre se requería el socket del cliente. Por esta razón, creamos funciones que interactúan directamente con los controladores de viajes y usuarios. Como resultado, tuvimos que crear dos variables estáticas para estos controladores, permitiendo un acceso global tanto para los servicios brindados a TcpClient como para los del administrador del servidor.
- **Usuario Específico para el Administrador**: Se creó un usuario específico que corresponde al administrador del servidor, diferenciando claramente sus permisos y funciones.

Inicialización y Funcionamiento del TcpServer

- **Inicialización de Colas**: Las colas se inicializan al comienzo del TcpServer, preparando el sistema para gestionar las comunicaciones desde el inicio. Tuvimos que crear 3 colas, 2 para los viajes y 1 para los usuarios, con un exchange directo para cada una. Esto lo hicimos ya que nos pareció lo más fácil de implementar, permitiendo enviar mensajes a colas específicas basándose en claves de enrutamiento
- **Servidor de Estadísticas**: Este servidor se mantiene en un bucle constante, escuchando RabbitMQ con dos hilos por cada cola y utilizando un token de cancelación para saber cuándo finalizar.

Gestión de Datos y Persistencia

- **DTO (Data Transfer Objects)**: Añadimos una carpeta DTO para estructurar y facilitar el envío de datos a RabbitMQ, permitiendo especificar qué datos deben enviarse o persistir en otros servidores.
- **MemoryDatabase**: Creamos una clase estática MemoryDataBase que contiene usuarios, viajes y reportes generados y persistidos, proporcionando un almacenamiento temporal accesible para todas las instancias.

Arquitectura del Servidor

- **Biblioteca de Clase para TcpServer**: El servidor TcpServer se convirtió en una biblioteca de clase en lugar de una aplicación de consola. Ahora se inicia desde el servidor gRPC, creando una instancia Singleton de TcpServer para que los servicios gRPC puedan acceder fácilmente a sus funciones.

Reportes y Gestión de Viajes

- **Clase Reporte**: Desarrollamos una clase para gestionar los reportes, ya que consideramos necesario persistir los distintos reportes creados. Según los requerimientos, es necesario consultar el estado de los reportes para saber si están listos o no. Por esta razón, al crear un reporte, se muestra al usuario el ID correspondiente, con el cual podrá consultar el estado del mismo. Para completar los reportes, cuando se persiste un viaje o se lee desde RabbitMQ, recorremos la lista de reportes en MemoryDataBase y verificamos si el reporte está listo. En caso de que no esté listo, el viaje se agrega a la lista de viajes del reporte.
- **Funcionalidad de los proximos N viajes:** Para implementar esta funcionalidad, al solicitar los próximos N viajes en tiempo real, comenzamos a leer de la cola de viajes creada específicamente para el servidor del administrador. Además, eliminamos los viajes leídos anteriormente para dar la impresión de que se está leyendo de forma asincrónica y en tiempo real.

Manejo de Errores

- **Excepción para Fallos de Conexión con RabbitMQ**: Añadimos una excepción específica para gestionar los fallos de conexión con RabbitMQ, asegurando que el sistema pueda manejar estos errores de manera adecuada.

Guia de Instalación:

Para probar todas las funcionalidades es especialmente necesario tener .net y docker.

RabbitMQ:

Necesitamos esta dependencia para lograr la comunicación entre nuestras aplicaciones. Para ello podemos levantar un contenedor con el siguiente comando:

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672
rabbitmq:3.9-management-alpine
```

Colleccion de Postman:

Este módulo de la documentación sirve exclusivamente para aclarar que en el documento entregado existe una colección de postman que tiene como finalidad facilitar la corrección del servicio de estadísticas.