Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio 2 - Programación de Redes

Entrega Final

Documentación

Brahian Peña - 267633

Septiembre

2022

Índice

1. Descripción General del Trabajo

En este documento se describe la solución implementada, la cual es un sistema completo de gestión de perfiles de trabajo. Este cuenta con varios modulos, los cuales pueden ser accedidos mediante la red para realizar diferentes tareas (manejo de perfiles, tareas administrativas, logging de mensajes, etc).

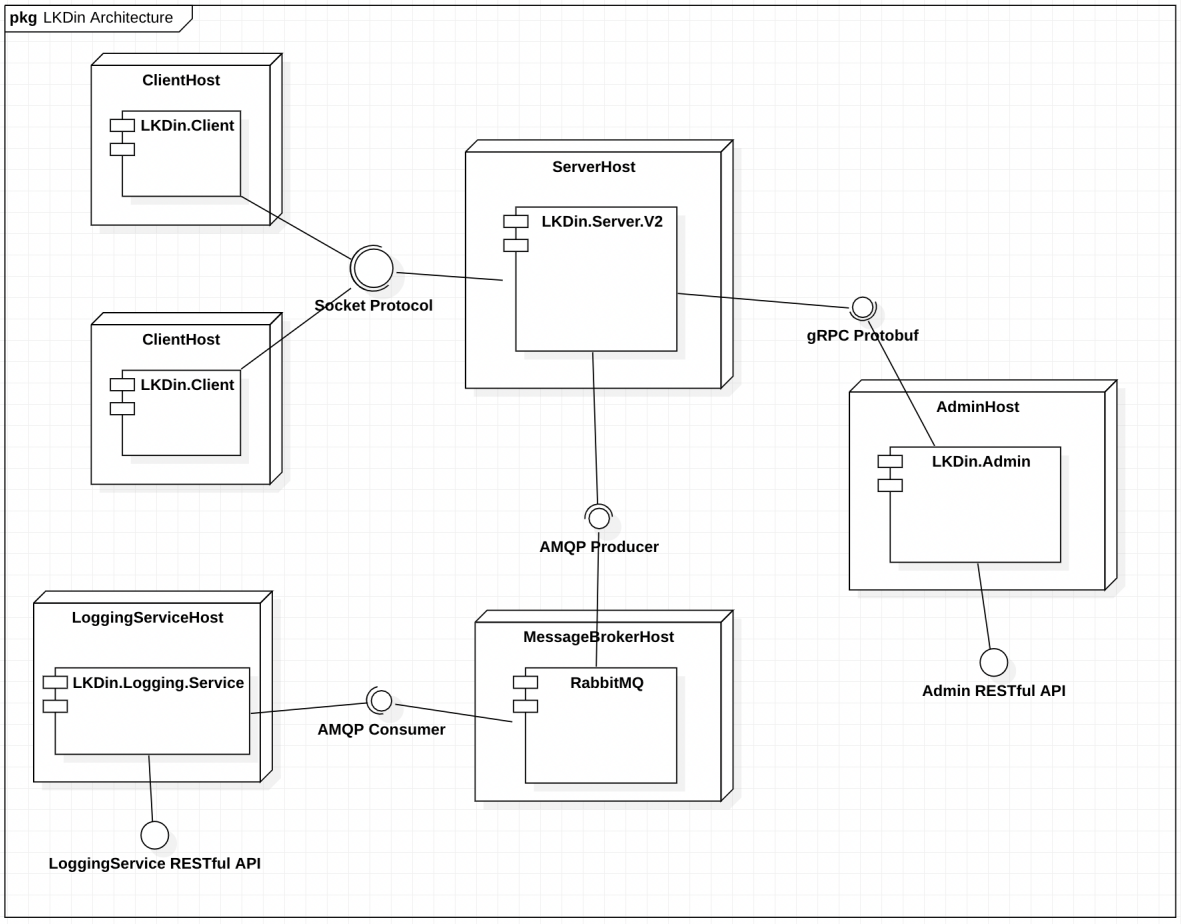
El servidor principal del sistema cuenta con la capacidad de persistir información en un sistema de archivos de texto plano. También puede recibir y manejar conexiones concurrentes (mediante TCPListener / TCPClient – para la comunicación - y Async Tasks – para el manejo de concurrencia). Cualquier tercero que implemente el protocolo de mensajería puede comunicarse con el servidor de manera full dúplex. Ademas este expone un servicio gRPC el cual es consumido por una aplicación administrativa (la cual expone una API Rest). A su vez, para ayudar a monitorear y detectar posibles errores en la aplicación principal, este envia sus logs a una cola de mensajeria (mediante AMQP), la cual es manejada y persistida por RabbitMQ. Esta cola de mensajeria es consumida por un servicio de logs que centraliza el manejo de estos y los permite consumir mediante una API Rest.

Gracias al diseño implementado la mayor parte de la lógica se encuentra en el servidor principal.

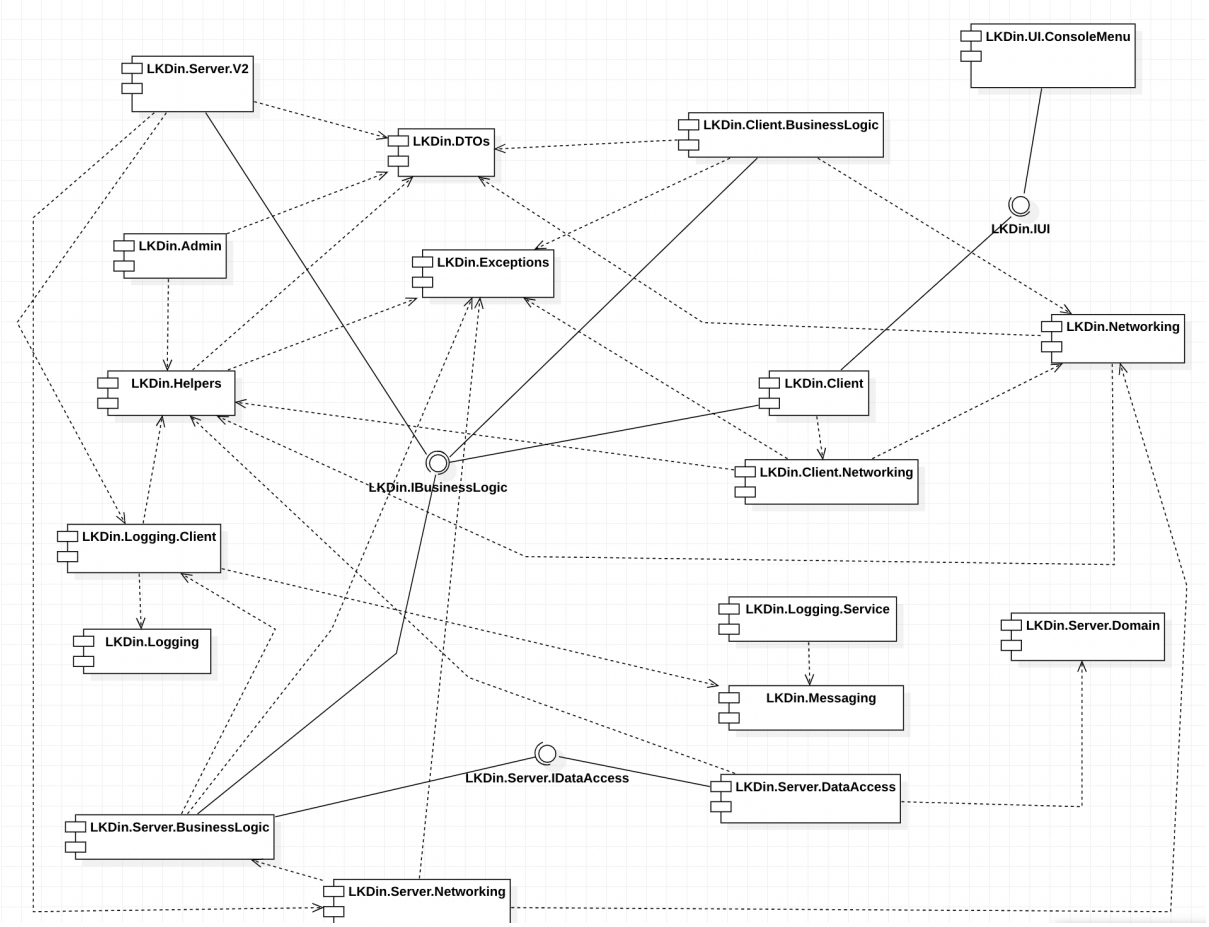
Al momento de escribir este documento, se cree haber implementado todas las funcionalidades solicitadas.

Se utilizó .NET 6 para construir la solución.

2. Diagrama de Arquitectura del Sistema



3. Breve Descripción de los Paquetes



Se adicionaron a la entrega anterios los siguientes paquetes / proyectos:

LKDin.Messaging

Incluye clases wrappers que permiten consumir y publicar mensajes a un servicio de mensajera RabbitMQ (utilizando el protocolo AMQP).

LKDin.Logging

Contiene clases que permiten estandarizar el proceso de envio y recepcion de logs entre las distintas aplicaciones involucradas.

LKDin.Logging.Client

Contiene las funcionalidades necesarias para loggear mensajes y transportarlos al servicio de mensajeria.

LKDin.Logging.Service

Incluye el servicio consumidor de logs, el cual centraliza y almacena los logs. Ademas, ofrece una API Rest para consultar los logs almacenados.

LKDin.Server.V2

Contiene la aplicación servidor principal, la cual contiene la mayor parte de la lógica del sistema. Además expone diferentes interfaces (gRPC y un protocolo definido para comunicación mediante TCP sockets) para que otras aplicaciones se puedan comunicar con el. Notar que este paquete reemplaza el paquete **LKDin.Server**

LKDin.Admin

Contiene el servidor administrativo, el cual interactua con el servidor principal mediante gRPC. Ademas, este servidor admin expone una API Rest por la cual se puede interactuar con el.

3. Análisis, Justificación del Diseño y Cambios

3.1 Estructura de la solución

Se mantiene la idea planteada en la primer entrega (ver mismo punto de esa documentación).

3.2 Serialización de datos

Adicionalmente a lo planteado para la primer entrega (ver mismo punto de documentación de primer entrega), se utilizó JSON como formato elegido para solicitar y servir información a través de los servicios RESTful.

3.3 Mecanismos de acceso a datos

Ver mismo punto de documentación de la primer entrega.

3.4 Comunicación entre aplicaciones

Adicionalmente a lo planteado en la primer entrega, el servidor principal ahora también expone un servicio gRPC el cual es consumido por el servidor admin. Luego, también el servicio de logs y el servidor principal se comunican con RabbitMQ para consumir y publicar mensajes de log. Tanto el servidor admin como el de logs, exponen una capa de servicios RESTful.

Se eligió gRPC como la tecnología para comunicar el servidor principal con el servidor administrador ya que nos permite hacer llamadas a procedimientos de manera remota de una forma rapida y prolija.

A su vez, se decidió utilizar una cola de mensajeria para transmitir los logs del servidor, así el servicio de loggeo los consume a medida que va pudiendo, y no es necesario que este esté andando para que el servidor principal también funcione (se genera un bajo acoplamiento entre las aplicaciones).

Luego, se eligio REST como el estandar para construir las WebAPIs que exponen tanto el servidor de logs como el admin, ya que permite que otras aplicaciones se puedan integrar facilmente a estás.

3.5 Mecanismos de control de concurrencia

Ya que cada cliente se maneja en un Task asincronico, es necesario mantener el control de la concurrencia para asegurar la integridad de los datos. Para ello se utilizan candados de mutua exclusión (*locks*) a la hora de acceder a los archivos de datos. También a la hora de crear nuevas entradas en esos archivos, se verifica que el identificador ya no exista. Notar que los metodos que acceden a los archivos no tienen llamadas a operaciones asincronicas dentro, por lo que **no** es necesario usar candados asincronicos en vez de los *locks*.

3.6 Manejo de excepciones

Adicionalmente a lo planteado en la primer entrega, y ya que se agregan nuevas capaz de servicios, se decidió implementar un *ErrorHandlerInterceptor* para el servidor gRPC, el cual captura las excepciones lanzadas por la aplicación y las normaliza para ser enviadas mediante gRPC al cliente. También se hizo uso de los códigos de error internos definidos por gRPC: Ver gRPC Status Codes. Del lado del servidor administrador, un filtro de excepciones las captura y las convierte a códigos de estado HTTP.

3.7 Configuración de la aplicación

A diferencia de lo planteado en la primer entrega, se decidió estandarizar el uso de archivos JSON para la configuración de las diferentes aplicaciones. Está configuración puede ser modificada sin necesidad de recompilar ningún modulo. Ejemplo de configuración de el servidor principal:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Kestrel": {
    "EndpointDefaults": {
      "Protocols": "Http2"
    }
  },
  "SOCKET_SERVER_IP": "127.0.0.1",
  "SOCKET_SERVER_PORT": 14000,
  "ABS_APP_DATA_PATH": "/LKDin_Server",
  "ABS_DOWNLOADS_PATH": "/LKDin_Server_Downloads",
  "BROKER_HOSTNAME": "localhost",
  "BROKER_PORT": "5672",
  "BROKER_USER": "admin",
  "BROKER_PASS": "admin",
  "BROKER_EXCHANGE": "lkdin.logging",
  "BROKER_QUEUE": "server.logs"
}
```

4. Protocolo de Comunicación

No se realizaron modificaciones sobre el protocolo planteado en la primer entrega.

5. Instalación de la Aplicación

Para instrucciones de instalación, ver punto 5 de la documentación de la primer entrega.