Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio 1 - Programación de Redes

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 de gestión de perfiles de trabajo, siendo la aplicación principal un servidor que puede ser accedido mediante la red utilizando un cliente propietario.

La implementación cuenta con la capacidad de persistir información en un sistema de archivos de texto plano. También puede recibir y manejar conexiones concurrentes (mediante TCP Sockets – para la comunicación - y Multihreading – para el manejo de concurrencia). Cualquier tercero que implemente el protocolo de mensajería puede comunicarse con el servidor de manera full dúplex.

Gracias al diseño implementado la mayor parte de la lógica se encuentra en el servidor, y el módulo de interfaz de usuario es el mismo para las dos aplicaciones (cliente y servidor).

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

2. Breve Descripción de los Paquetes

LKDin.Server.Domain

Este paquete contiene las entidades de la aplicación, lo que formaría parte del núcleo de la aplicación.

LKDin.DTOs

Contiene los Data Transfer Objects que serán enviados a través de la red, son compartidos tanto entre cliente como servidor.

LKDin.Server.IDataAccess

Contiene las interfaces de los repositorios que permiten consultar y persistir información.

LKDin.Server.DataAccess

Este proyecto es una librería de clases que ofrece los repositorios que implementan sus respectivas interfaces de IDataAccess. También incluye la clase *DataManager*, la cual se encarga de interactuar con el File System para el manejo de los datos de la aplicación.

LKDin.IBusinessLogic

Este paquete contiene las interfaces que implementan las clases de la lógica de negocio de la aplicación. Tanto servidor como cliente dependen de este paquete.

LKDin.Server.BusinessLogic

Este proyecto incluye las clases que implementan las interfaces de la lógica de negocios en el servidor. Son el principal entry-point de los datos.

LKDin.Exceptions

Contiene excepciones personalizadas de la aplicación, también se comparte entre cliente y servidor.

LKDin.Helpers

Este paquete contiene utilidades y extensiones para la serialización, manejo de assets (principalmente imágenes) y manejo de configuración. La mayoría son clases estáticas.

LKDin.IUI

Incluye las interfaces que deben implementar las clases que ofrecen la utilidad de interactuar con los usuarios de la aplicación.

LKDin.UI.ConsoleMenu

Este paquete incluye las clases que se encargan de interactuar con el usuario, tanto el menú de usuario de consola, como los comandos / opciones disponibles. Este paquete es utilizado tanto por el cliente como por el servidor.

LKDin.Networking

Contiene las clases e interfaces que se encargan de manejar las conexiones entre sockets, convertir, transmitir y recibir información. También incluye las constantes del protocolo.

LKDin.Server.Networking

Contiene las clases que manejan las conexiones mediante socket al servidor.

LKDin.Client.Networking

Contiene las clases que manejan las conexiones mediante socket del cliente al servidor.

LKDin.Client.BusinessLogic

Contiene la lógica de negocio del cliente, la cual se encarga principalmente de enviar mensajes al servidor, usando las clases de *LKDin.Networking*.

LKDin.Client

Contiene la aplicación cliente.

LKDin.Server

Contiene la aplicación servidor.

3. Análisis y Justificación del Diseño

3.1 Estructura de la solución

Como se puede ver, la solución está separada en diferentes proyectos, estos agrupan funcionalidades relacionadas, que, a la hora de compilarlos, se pueden hacer por separado. Esto nos permite distribuir de mejor forma la aplicación, mejorando la capacidad de reusó de los diferentes módulos.

Otra cosa para destacar es que las interfaces tienen sus propios paquetes, esto nos permite aplicar el principio de inversión de la dependencia, para que los módulos de alto y bajo nivel, solo dependan de interfaces bien definidas y no de implementaciones concretas.

Cada paquete tiene su propia responsabilidad y estás no están mezcladas entre paquetes, como se comentaba en la descripción de los paquetes.

También se intentó replicar el patrón de inyección de dependencias para inyectar instancias de clases de lógica de negocio en los comandos de la UI disponibles, permitiendo el reusó del módulo de la UI.

3.2 Serialización de datos

Para almacenar la información de los archivos y transmitirla se necesita convertir los objetos a *string*. Para esto se implementó la clase estática *SerializationManager* la cual ofrece métodos para transformar información de objetos a *string* y viceversa. Usa principalmente funcionalidades que hacen uso de Reflection para obtener los nombres de las propiedades y asignar los valores a las propiedades.

3.3 Mecanismos de acceso a datos

A la hora del acceso a datos, se tuvo en cuenta la utilización del patrón repositorio. Así el acceso a datos queda encapsulado en un solo paquete, mejorando la mantenibilidad y seguridad del sistema.

De todas formas, este diseño puede ser mejorado, ya que se puede generar un repositorio base que sea abstracto y después hacer que las diferentes implementaciones del repositorio hereden todos los métodos ya implementados de las funcionalidades básicas (CRUD), para así evitar repetir código.

También se encapsuló el acceso a los archivos de datos dentro de la clase *DataManager*, estas son usadas por los repositorios.

Ver punto 3.4 que informa sobre acceso concurrente a estos archivos.

3.3 Recepción y envío de mensajes

Como se comentaba, el servidor acepta conexiones TCP mediante socket en un IP y puerto asignados. Cuando un nuevo cliente se conecta, se crea y se le asigna un nuevo hilo que será el encargado de atender a ese cliente en concreto. Cada cliente puede utilizar la clase *NetworkDataHelper* para enviar y recibir mensajes.

3.4 Mecanismos de control de concurrencia

Ya que cada cliente se maneja en un hilo diferente, es necesario controlar 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.

3.5 Manejo de excepciones

Para este caso, se decidió crear excepciones personalizadas, las cuales permiten un manejo de errores simplificado.

Estas se capturan generalmente al nivel de comando de interfaz de usuario, excepto cuando son lanzadas a partir de una llamada del cliente al servidor, en ese caso se captura a nivel del manejador de conexiones del servidor, se serializa, se envían por el socket al cliente, el cual las recibe y las vuelve a lanzar.

3.6 Configuración de la aplicación

La configuración básica de las aplicaciones se realiza mediante un archivo de configuración llamado *App.config,* el cual se encuentra en la misma carpeta donde se encuentra el ejecutable. Las variables de configuración disponibles son las siguientes:

- ***ABS_DOWNLOADS_PATH***: Carpeta en la que se guardaran las descargas.
- ***SERVER_PORT***: Puerto que se le asignará el servidor, o bien para el caso del cliente el puerto en el que escucha el servidor.
- ***SERVER_IP***: IP que se le asignará el servidor, o bien para el caso del cliente la IP asignada al servidor dentro de la red.
- ***ABS_APP_DATA_PATH:*** Lugar donde se almacenará la información de la aplicación (utilizado principalmente por el servidor)

Ejemplo de configuración:

```xml
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

    <appSettings>

        <add key="SERVER_IP" value="192.168.1.150"/>

        <add key="SERVER_PORT" value="14000"/>

        <add key="ABS_APP_DATA_PATH" value="/LKDin"/>

        <add key="ABS_DOWNLOADS_PATH" value="/LKDin_Server_Downloads"/>

    </appSettings>

</configuration>
```

4. Protocolo de Comunicación

A continuación, se especifica el protocolo propietario de comunicación implementado:

- Protocolo orientado a caracteres.
- Implementado sobre TCP/IP.
- Los valores deberán ir alineados a la derecha, los caracteres de relleno deberán tener el valor 0.
- El campo *HEADER* (que incluye *CMD* y *LENGTH*) tendrá largo fijo (a su vez cada campo dentro de este tiene largo fijo, ver la tabla siguiente). Los datos (el cuerpo del mensaje) tendrá largo variable, según el valor indicado en el header *LENGTH*.

Formato general:

Nombre del Campo	**HEADER**	**DATOS**
Valores disponibles	CMD | LENGTH	Variable
Largo	Total 26 (CMD: 4 | LENGTH: 10 + Largo De Claves)	Variable
Ejemplo	CMD=0001|LENGTH=0000000117	Name(string)=Lionel Messi|Password(string)=$2a$11$wOeqqPMzYUKHT3O2w43Gn.gwtRhndCP88ZZAoaOtSctwaf61c21bC|Id(string)=1

5. Instalación de la Aplicación

La aplicación no requiere instalación, simplemente se puede ejecutar el archivo ejecutable de la carpeta que la contiene para ejecutarla. Al iniciar, esta muestra la carpeta del sistema de archivos donde se almacena la información y donde se pueden encontrar los archivos de configuración.