Universidad ORT Uruguay

Facultad de Ingeniería

Programación de redes
Obligatorio 1

Jimena Sanguinett(228322)
Joselen Cecilia(233552)

Entregado como requisito de la materia Programación de redes

27 de abril de 2021

Declaraciones de autoría

Nosotras, Joselen Cecilia y Jimena Sanguinetti, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Programación de redes ;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

1. Introduccion

Se construye un sistema formado por dos aplicaciones de consola con un menú cada una, cuenta con un servidor en el que se deben guardar datos de posts de usuarios los cuales contienen temas y archivos. Además cuenta con un cliente para dicho servidor que se encargará de permitir interactuar a los clientes con el sistema.

El sistema cumple con las siguientes funcionalidades:

Del lado servidor:

-Aceptar pedidos de conexión de un cliente
-Mostrar los clientes conectados
-Listado de temas
-Listado de posts por tema
-Ver un post especifico
-Ver archivo asociado a un post especifico
-Ver tema con más posts
-Ver listado de archivos

Del lado del cliente:

-Conectarse y desconectarse al servidor
-Alta de tema
-Baja y modificación de tema
-Alta de post
-Baja y modificación de post
-Asociar y desasociar post a un tema
-Subir archivos relacionados con el post

2. Alcance

Cuando se inicia la aplicación cliente se presenta un menú con las distintas opciones que puede seleccionar el usuario del lado del cliente. Las opciones se dividen por Post, Tema, asociar Archivo y buscar post; dentro de post, se encuentran diferentes opciones como agregar, editar, eliminar y asociarles/desasociarles un tema. Al agregar un post, este se guarda en una lista en MemoryRepository junto al resto de los posts, al eliminarlo se borra de esta misma lista y al editarlo, se borra el actual y se vuelve a agregar con los datos actualizados. En caso de que el post esté en uso por otro cliente, al intentar eliminarlo o actualizarlo, no será posible. Por último, dado un post, el cliente puede tanto asociarle como desasociarle un tema existente en el sistema; en caso de que se desasocie un tema, se debe asociarle uno nuevo.

Dentro de los temas también es posible agregar, editar, eliminar con el mismo comportamiento que los posts. Una vez agregado un tema, este puede ser luego asociado a un post.

Con respecto a los archivos, se debe elegir un post para asociarle a un archivo; el archivo hereda los temas del post.

Por último, en la opción de buscar un post, se selecciona uno de la lista y se muestran todos sus datos.

Por otro lado, cuando se inicia la aplicación servidor, también se presenta un menú pero con las opciones que se pueden seleccionar del lado del servidor. Estas se dividen por listar clientes, Post, Tema y Archivo; en la primera opción se muestran todos los clientes que están conectados al servidor, mostrando la ip, el puerto y fecha y hora en que se conectó. Con respecto a los Posts, nuevamente se muesta un menú que presenta las opciones por las cual se desea filtrarlos; es posible listarlos ordenados según la fecha de creación, según un tema que se seleccione o por ambos filtros. Además, se puede seleccionar un post en particular y se muestran todos sus datos. También es posible seleccionar un post para el cual se muestran todos los datos del archivo asociado.

Dentro de los temas es posible mostrar un listado de todos los que existen dentro del sistema. Además se puede mostrar el tema que tiene mayor cantidad de posts asociados. En caso de existir más de uno con la mayor cantidad, se muestran todos ellos.

Por último, también es posible mostrar una lista de todos los archivos; una opción es mostrar la lista completa pero también se puede filtrar por tema, por fecha de creación, por nombre o por tamaño.

3. Descripción de la arquitectura

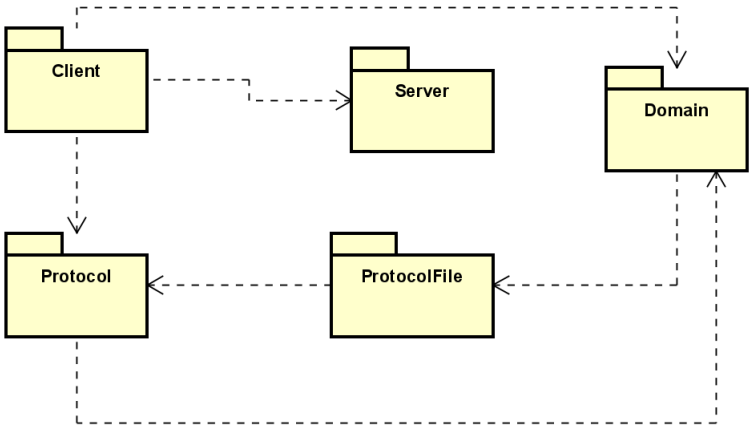
Hilos:
Del lado del servidor fue necesario implementar el uso de hilos; decidimos utilizar dos hilos diferentes, uno para manejo del menú del lado del servidor y el otro hilo es para aceptar las conexiones de los distintos clientes, el cual también se utiliza para manejar las solicitudes que realiza el cliente mediante su propio menú.

Sockets:
Para establecer el enlace de red para que el cliente se pueda comunicar con el servidor y viceversa, fue necesario el uso de sockets. Estos nos permiten mantener una comunicación bidireccional para enviar y recibir data. En nuestro caso, esta data son paquetes con información, la misma es explicada más adelante en la definición del protocolo.

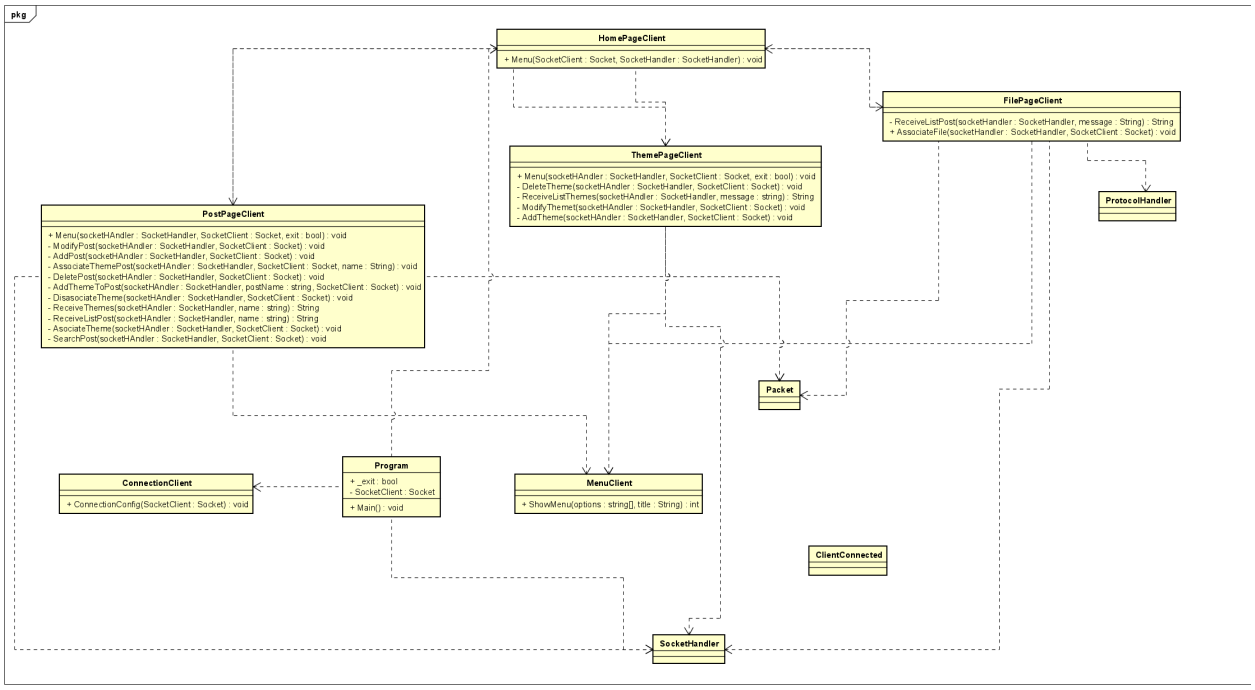
Stream:
Par poder enviar y recibir archivos fue necesario el uso de la clase NetworkStream de modo que se pueda acceder a los métodos y propiedades de la clase Stream (clase padre) que se utilizan para facilitar las comunicaciones de red. Una vez que se obtiene una instancia de esta clase, se llama al método Write para enviar data al servidor, mientras que el servidor llama al método Read para poder recibir la data que llega. Ambos métodos se bloquean mientras se esta realizando la operación; la operación de lectura se desbloquea cuando los datos han llegado y están disponibles para ser leídos.

4. Diseño detallado de cada uno de sus componentes

El diseño de nuestra aplicación se basa en varias capas las cuales tienen diferentes propósitos cada una, la división de las mismas fue definida justamente por la función que cumple dentro de la aplicación. Así mismo, estas capas también se subdividen, principalmente para aumentar la organización y facilitar el entendimiento. Estas son:



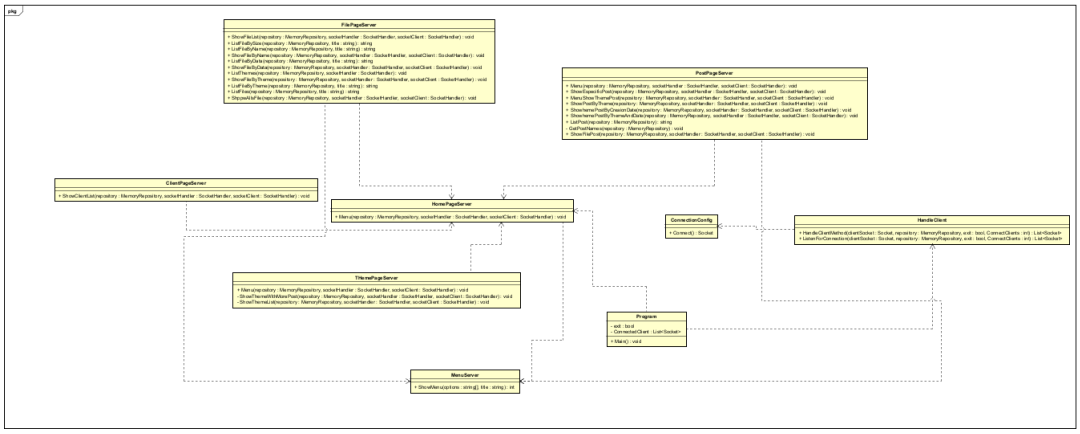
1. Client:



En esta capa podemos encontrar al cliente que se comunicará con el servidor, este es quien manda las diferentes requests correspondientes a las diferentes acciones que se pueden realizar. Es posible establecer comunicación entre el servidor y cuantos clientes se quieran, por lo tanto lo que se hace es, a cada cliente asignarle su socket y su endpoint correspondientes para luego poder identificarlo y así manejar las distintas respuestas que recibe de las distintas requests del lado del servidor.

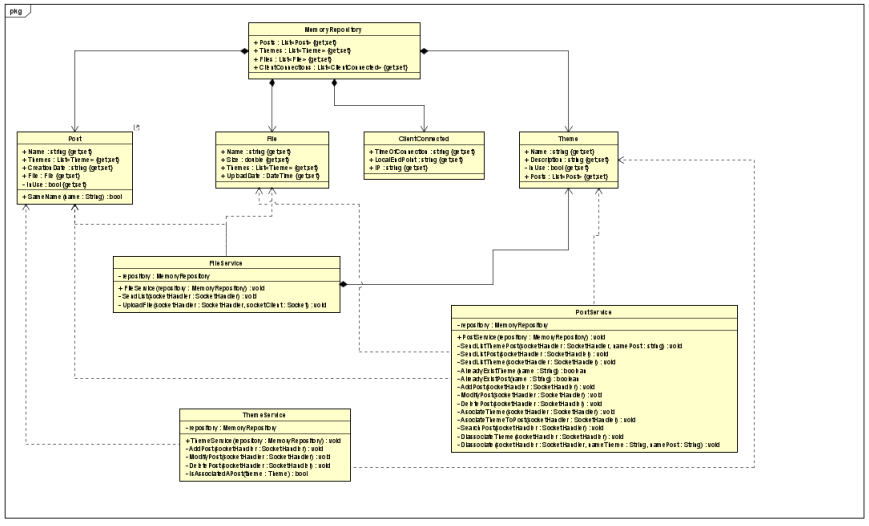
Dentro del cliente podemos ver tres principales subdiviciones, contamos con la clase Program, en esta se llama a la calse connectionConfig donde se hace referencia a los puertos e IPs definidas en el appConfig para poder establecer la conexión; además se define el socket y su endpoint. Por otro lado tenemos un MenuClient donde se define como se va a mostrar el menú con las distintas opciones de las funciones que puede solicitar el cliente; decidimos que el menú se pueda controlar por el teclado de modo que sea más amigable para el usuario así como también para nosotras, dado que fue más sencillo de validar. Por último contamos con una carpeta llamada Pages, en esta podemos encontrar la lógica, dividida en cuatro clases diferentes: FilePageClient, HomePageClient, PostPageClient y ThemePageClient. Cada una de estas contiene todas las funcionalidades respecto a cada una de las entidades en cuestion (Post, File y Theme), además de HomePage la cual muestra el menú propio del cliente en el cual se delegan las diferentes tareas a las otras Pages; en estas se realizan las requests hacia el servidor.

2. Server:



El diseño del lado del servidor es muy similar al del cliente; podemos encontrar la clase Program, MenuServer, la carpeta con las distintas pages, appConfig, connectionConfig, las cuales se encargan de la configuración de las conexiones. Por último tenemos una clase HandleClient. Esta clase es la que se encarga de manejar las request realizadas por cualquier cliente; podemos encontrar todas las diferentes funciones que el cliente puede solicitar y esta delega la implementación de la misma a los Service de cada objeto que se mencionan en el paquete del dominio.
La clase Program cumple la función de aceptar y manejar a los distintos clientes, además es quien se relaciona con el repositorio en memoria para guardar los clientes conectados con sus respectivos sockets. Este se encarga de lanzar un hilo donde se muestra el menú del servidor.

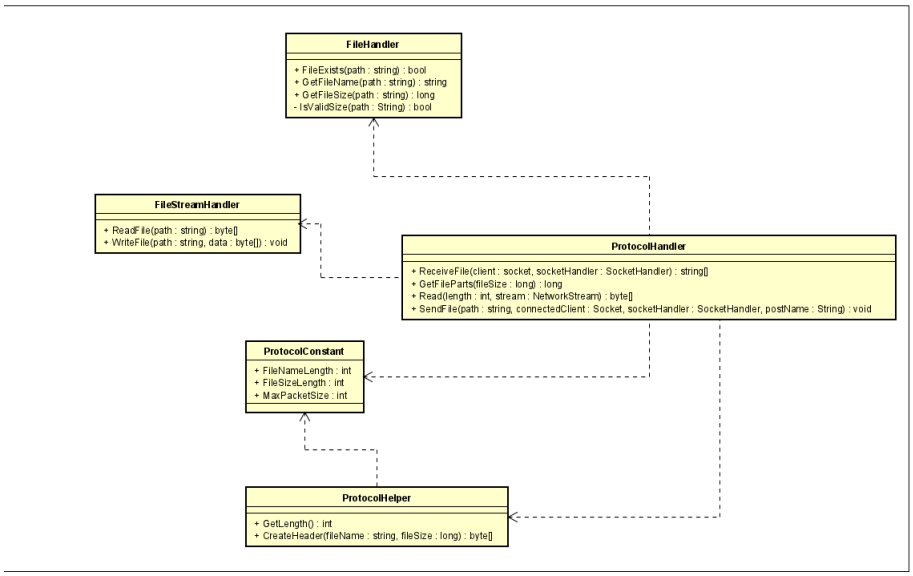
3. Domain:



En el dominio podemos encontrar las distintas entidades en cuestión: File, Theme, Archive y Post; además contamos con la clase ClientConnection que contiene la ip, el puerto y la hora de conexión de cada cliente, la cual es utilizada para poder listar todos los clientes conectados al servidor, mostrando esos tres datos de cada uno. Por último tenemos el repositorio MemoryRepository que contiene las listas

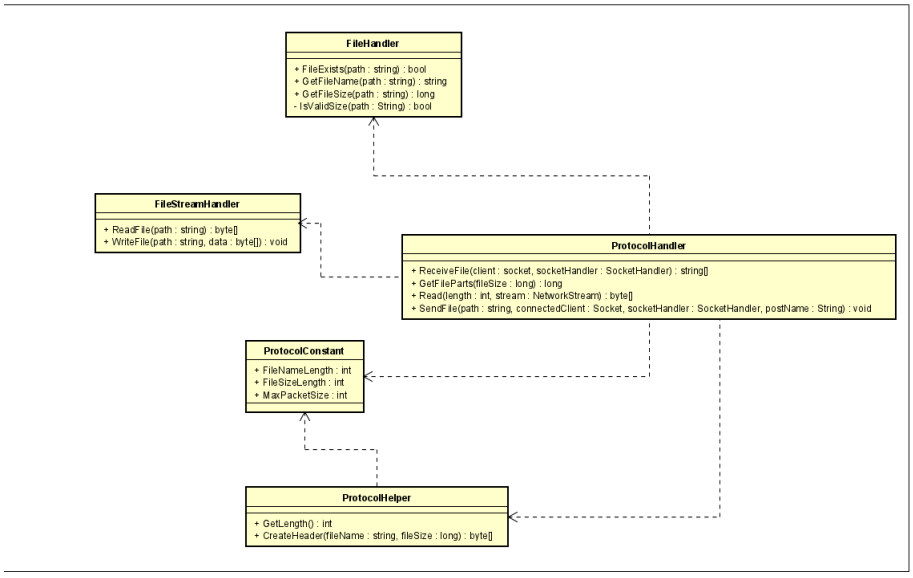
que deseamos guardar en memoria: Posts, Themes, Files y ClientConnections. Además contamos con una carpeta Service que contiene un Service por cada entidad, en cada uno de estos encontramos una instancia del memoryRepository ya que estas clases se utilizan para el manejo de las listas en memoria.

4. Protocol:



En esta capa se maneja el envío de datos mediante el protocolo definido por nosotras. Contamos con el recieve y send de los packages; estos son definidos dentro de ese paquete y están formados por un Header, un Command, un length y un Data, los cuales son explicados más adelante en la definición del protocolo. Por último se tienen las constantes que se utilizan para decidir el comportamiento frente a las requests.

5. ProtocolFile:



En esta capa podemos encontrar todas las especificaciones del protocolo para el envío y la recepción de archivos; para esto fue necesario definir las clases: FileHandler, FileStreamHandler, ProtocolHandler, ProtocolHelper y ProtocolSpecification.

Nota: en caso de querer ver los UML con mejor calidad, se encuentran adjuntados en la carpeta del obligatorio dentro de UMLs

5. Documentación de mecanismos de comunicación de los componentes de su solución

5.1. Protocolo

Para que el Servidor y el Cliente puedan comunicarse de forma que el servidor reciba las requests y envíe las respuestas para que luego el cliente las interprete, fue necesario definir dos protocolos, uno para el envío de strings (Protocol) y otro para el envío de archivos (ProtocolFile).

Protocol

Para el envío y recibo de la información, se crearon dos métodos SendPakg y ReceivePakg los cuales se encargan de enviar y recibir los paquetes con la información.

Estos paquetes están formados por un "header", un çommand", un "lengthz un "data"; el header nos indica si el mensaje se manda por una response o una request (RES/REQ), este tiene un largo fijo de3 bytes; el command es el número del requerimiento, que me indica cual funcionalidad debo llamar y tiene un largo fijo de 2 bytes, si el largo es de una cifra se rellena con ceros a la izquierda (ejemplo: 02).
El length me indica el largo de la data y se le suman cinco bytes(header+command) y tiene un largo fijo de 4, al igual que el command si la cifra del length es menor a 4 se rellena con ceros a la izquierda (ejemplo: 0012, indica que la data es de largo 12)
Por último, la data es un string que contiene toda la información que se esta enviando concatenada mediante #. Para completar el paquete.

Antes de mandar la información es necesario crear el Packet mientras que una vez que se recibe se debe hacer el proceso inverso para que se pueda interpretar la data del mismo. Este proceso inverso implica separar, según los #, la data del Packet, lo que nos permite acceder a la información en sí.
En el receive primero se reciben, con la función receive de socket, el header, el commmand y el length almacenándolos en un array de largo 9 (3 para header,2 para command y 4 para el lengt), que luego se convierte en string para asignarlos al packet.command, packet.header y packet.length respectivamente.

Luego con ese length mas nueve (estos 9 son: 3 del header,2 del command y 4 del length) recibimos la data, el mensaje que se envió tambien con la funcion de socket receive almacenándola primero en un array de largo length y luego convirtiéndola en string para asignársela al packet.data.
En el send se recibe un packet, se junta toda la información del packet en un string y se manda usando el send del socket.

Nombre del campo	Header	Command	Length	Data
Valores	RES/REQ	1-12	0-9999	Data#data#...
Largo	3	2	4	Length

ProtocolFile

Para enviar y recibir archivos, se debe definir un FileNameLength, un FileSize y un MaxPacketSize; a partir de estos datos se define el paquete en el que se va a mandar. Este paquete se va a dividir en determinada cantidad de partes, dependiendo del MaxPacketSize ya que este es el tamaño máximo que puede tener.

Cuando el cliente envía el archivo utiliza el método WriteFile, invocado dentro del método SendFile, mientras que el servidor recibe el archivo llamando al método ReadFile invocado dentro del método RecieveFile. Dentro del envío y la recepción del archivo, también se incluye el envío de los datos del mismo (nombre, tamaño y Post al cual se va a asociar), como un paquete.

6. Justificación de las decisiones importantes de diseño e implementación.

A la hora de implementar la solución de nuestro sistema tuvimos que tomar ciertas decisiones con respecto al diseño, las mismas fueron:

1. Dividimos el proyecto en dos soluciones, por un lado tenemos al cliente y por el otro al servidor. Cada uno cumple un rol diferente y tiene por lo tanto funcionalidades diferentes, lo cual nos llevo a mostrar un menú independiente uno del otro.

2. Para poder transferir información tuvimos que definir dos protocolos diferentes, uno que se encarga de los archivos y otro de los paquetes que contienen mensajes. Decidimos separar la implementación, los elementos y el manejo de cada protocolo en dos proyectos separados. Esta decisión se basó en presentar una solución más ordenada, además de mantener bajo el acoplamiento entre ambos protocolos, ya que estos son independientes uno del otro.

3. Dentro del protocolo utilizado para enviar los paquetes de información, decidimos concatenar toda la data mediante #. Usamos este símbolo dado que es un char comúnmente utilizado para la concatenación de strings. Cada vez que se crea un paquete, la data que se envía ya se encuentra concatenada con este formato.

4. Al momento de enviar un paquete, como se explica en el protocolo correspondiente, en caso de faltar digitos para completar el length, o el command de la data que son constantes de largo dos, se utilizan ceros a la izquierda. Esto se hace en el constructor del paquete con la funcion padLeft.

5. Para que un cliente no pueda editar, ni eliminar un post o theme mientras otro cliente realiza alguna de estas mismas acciones sobre el mismo post, se agregó a las entidades post y theme un bool InUse, que indica false, si no se esta modificando ni eliminando y true en caso contrario. Para cumplir con dicho requisito se podría haber utilizado el objeto Lock para que se bloquee el uso de dicho post o theme, creemos que es una posible mejora a implementar para la segunda parte de este obligatorio. Notamos esta mejor solución sobre el final de la entrega, razón por la cual decidimos no cambiar lo que ya teníamos.

6. Tanto el menú de servidor como el menú de cliente, se dividen en tres subpaginas correspondientes a cada entidad, PostPage,FilePage y ThemePage agregándoles al final server o client dependiendo en que proyecto se usan.
Esto se implementó de esta manera para agrupar las funcionalidades que trabajaban sobre la misma entidad en una misma clase (misma page), la cuál luego se llama en el caso que corresponda desde el menú principal que se encuentra en la pagina HomePage.

7. Dentro de Protocol, definimos CommandConstants, una clase que contiene como constantes las diferentes opciones de cada funcionalidad. Estas se asocian a un número, el cual será utilizado luego para hacer los casos del switch que se encuentra en la clase HandleClient, del lado del servidor. Esto busca que sea más entendible el switch ya que se expone el nombre de la funcionalidad y no un simple número.

8. Como mencionamos anteriormente, contamos con una carpeta Services dentro del dominio, su función es agrupar las distintas todas las clases Service (FileService, PostService y ThemeService) las cuales se encargan del manejo de listas en memoria. Decidimos agruparlas dado que forman parte del dominio pero no son las entidades en sí, sino que cumplen otro rol dentro del sistema. Una decisión similar fue tomada dentro del cliente y dentro del servidor; todas las pages son agrupadas en una carpeta Pages nuevamente para facilitar el entendimiento del código y separarlo según funcionalidades.

9. Se crean dos clases ConnectionConfig, una para el cliente y un para el servidor, donde se configura las conexiones usando los puertos y las IPs que se encuentran en el App.config

7. Manejo de errores

Los errores de datos mal ingresados, o acciones que no se pueden realizar en determinado momento se encuentran validados del lado del servidor, en las clases services, estos indican el error con un mensaje que se muestra luego por consola en el lado del client.

Los errores de conexión perdida o de sockets se captura con try-catch en los lugares correspondientes y se lanzan mensajes que se muestran también por consola. De la misma forma actúa la validación del path del archivo, donde en caso de ser una ruta incorrecta se lanza una excepción que es luego controlada en el filePageClient donde se informa al usuario que la ruta es incorrecta y se pide que se ingrese una nueva.

8. Manual de usuario

Luego de abierta la solución, se debe configurar en el App.config la IP correspondiente a la computadora del usuario.

Después de esta acción se debe ejecutar la aplicación de consola Server y luego la aplicación de consola Client.

Una vez ejecutadas las dos aplicaciones, el usuario verá dos menús, en los cuales deberá seleccionar la opción que desee.

Para seleccionar la opción el usuario debe desplazarse por el menú con las teclas down y up del teclado y luego al llegar a la opción que desee deberá presionar enter. Según la opción que seleccione se le mostrará la página que corresponda.

En los casos que se muestren las opciones con índices, se debe ingresar el número indicado en cada opción para poder acceder a la funcionalidad.

9. Link al repositorio de GitHub

https://github.com/JoselenC/Obligatorio1-Programacion-De-Redes.git