

Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio I

Entregado como requisito del primer obligatorio de la materia Programación de Redes

Franco Ligerini - 196335

Juan Casanova - 244108

Docentes: Luis Barrague

2020

Índice

Alcance de la aplicación	3
Restricciones	3
Arquitectura	4
Definición del protocolo:	4
Dominio del servidor	6
Server:	6
Connection:	6
User, Photo y Comment:	7
Logs	7
Componentes	8
Los distintos componentes	8
Cliente del usuario	8
Server Instafoto	8
Server de logs	8
Cliente administrador	8
Common	8
Esquema de los componentes	9
Uso de la aplicación	10
Server Instafoto	10
Cliente usuario	11
Log server	11
Cliente administrador	12
Link Repositorio	13

Alcance de la aplicación

Para el desarrollo de la aplicación se utilizaron todas las tecnologías solicitadas en la letra del obligatorio (Sockets, Stream, Diseño e implementación de Protocolo, RPC y RabbitMQ). La construcción de una API REST, como pedía el obligatorio no fue posible desarrollarla, dado que ninguno de los integrantes del grupo curso la materia Diseño de Aplicaciones 2.

Como mencionamos antes se utilizaron todas las tecnologías solicitadas. El uso de sockets fue para la comunicación entre el servidor InstaFoto y los clientes que solicitaban acceso al servicio a su vez se utilizaron también para comunicar el servidor de logs con los clientes administrativos así estos son capaces de mostrar los logs que se registraron; Para el envío de mensajes a través de sockets se utilizó un protocolo propietario. En cuanto al manejo de tecnologías MOM en nuestro caso nos manejamos con RabbitMQ el cual nos permitió establecer una comunicación entre el servidor InstaFoto(publisher) y el servidor de logs(consumer). Finalmente para la parte de RPC utilizamos gRPC para comunicar el servidor InstaFoto con los clientes administrativos.

Restricciones

- Tanto el nombre de los usuarios como las contraseñas no pueden contener el signo de % para el registro o modificación de los mismos.
- La comunicación a través de sockets siempre se realiza por la dirección del localhost(127.0.0.1).

Arquitectura

Definición del protocolo:

Se toma la estructura básica del protocolo visto en clase a la cual se le agregan algunas características extras. Una de ellas es la consolidación del header con el body en una única clase llamada “CommandPackage”, la cual permite una fácil definición y obtención de los datos desde ambos extremos.

```
32 references | Juan Casanova, 29 days ago | 1 author, 1 change
public class CommandPackage
{
    private byte[] _direction;
    private byte[] _command;
    private byte[] _datalength;
    private byte[] _data;

    1 reference | Juan Casanova, 29 days ago | 1 author, 1 change
    public string Direction { get; set; }
    5 references | Juan Casanova, 29 days ago | 1 author, 1 change
    public int Command { get; set; }
    4 references | Juan Casanova, 29 days ago | 1 author, 1 change
    public int DataLength { get; set; }
    13 references | Juan Casanova, 29 days ago | 1 author, 1 change
    public String Data { get; set; }
```

Estructura de la clase “CommandPackage”

A su vez, se definió una clase estática llamada “CommandProtocol” la cual permite un simple envío y recepción de paquetes entre el cliente y el servidor, al igual que funcionalidades para la transferencia de listas.

Ejemplo de uso del protocolo:

```
CommandPackage package = new CommandPackage(HeaderConstants.Request, CommandConstants.SendPicture, name);
CommandProtocol.SendCommand(socket, package);
```

```
CommandPackage package = CommandProtocol.RecieveCommand(client);
```

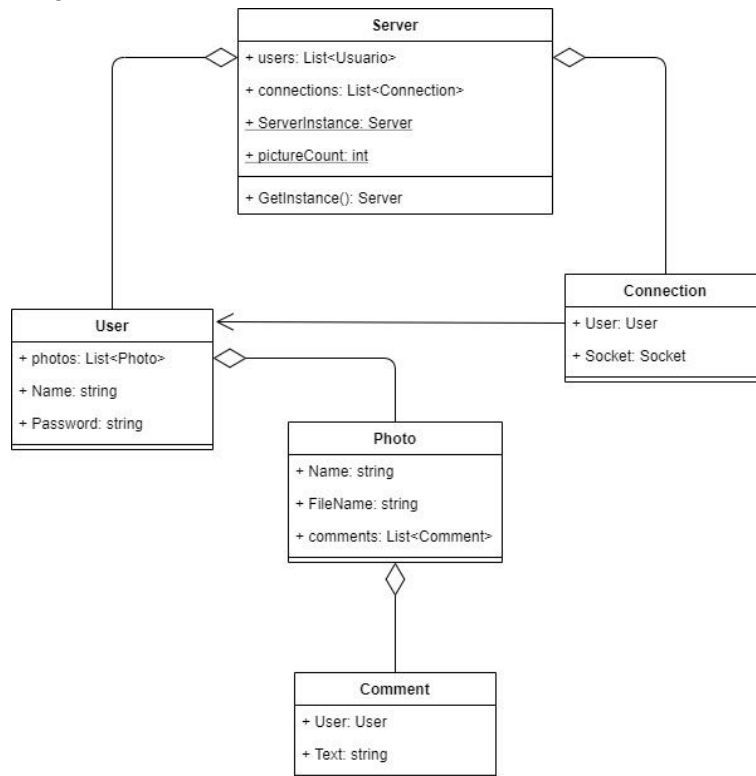
Se utiliza a su vez una librería de comandos ubicados en la clase “CommandConstants”, permite el simple envío de los comandos deseados manteniendo un código legible y mantenible.

```
public class CommandConstants
{
    public const int Exit = 0;
    public const int Login = 1;
    public const int Register = 2;
    public const int RequestLoggedUser = 3;
    public const int FinishSendingList = 4;
    public const int ACK = 5;
    public const int UserList = 6;
    public const int SendPicture = 7;
    public const int PictureList = 8;
    public const int CommentList = 9;
    public const int NewComment = 10;
    public const int Error = 99;
}
```

Por otro lado se definió un protocolo de transferencia de archivos basado en el visto en clase, el cual archiva de forma automática las fotos subidas asignándoles un nombre de archivo único, esto permite la existencia de múltiples fotos con el mismo nombre en el sistema, sin la posibilidad de que se den conflictos.

Dominio del servidor

Diagrama de clases simplificado



El dominio consta de cinco clases, las cuales están encargadas de almacenar en memoria la información del servidor y del manejo de los clientes.

Server:

La clase server es la encargada de mantener una referencia de toda la información del sistema, debido a que solo debe existir una instancia de dicha clase, se utilizó un patrón singleton para asegurar dicha premisa y para el fácil acceso por parte de otros componentes.

Sus responsabilidades incluyen la búsqueda y creación de nuevas conexiones con los clientes, manejo de usuarios y cerrar todas las conexiones cuando se lo desea.

Connection:

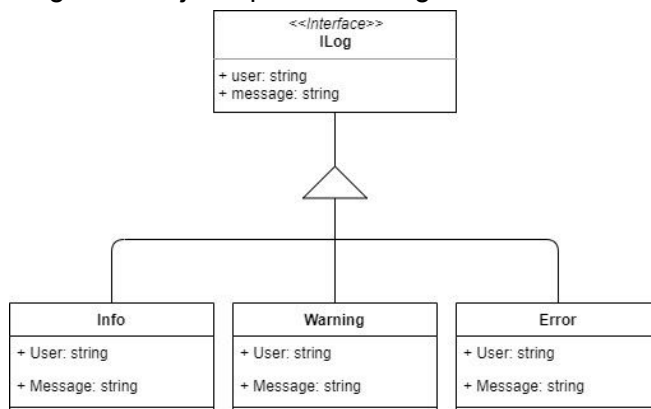
La clase connection permite el fácil manejo de todas las conexiones activas; incluye tanto el socket utilizado en la conexión, como el usuario que está haciendo uso de la misma. Cada una de estas tiene un thread encargado de enviar y recibir los comandos enviados por los clientes, con el fin de permitir un manejo paralelo de todas las conexiones.

User, Photo y Comment:

Dichas clases permiten almacenar toda la data respectiva de los usuarios (nombre, contraseña, fotos subidas, comentarios de dichas fotos y comentarios realizados).

Logs

Diagrama de jerarquía de los logs.



El servidor de Instafoto y el servidor de logs están conectados por una única queue de RabbitMQ, mediante el cual se transfieren los logs de forma simple mediante la serialización de los logs en formato Json.

Ejemplo de publicación de un log:

```
ILog log = new Info();
log.User = connection.User.Name;
log.Message = "Usuario solicita lista de usuarios";
Server.GetInstance().LogAction(log);
```

Json creado:

```
{
  "$type": "Common.Logs.Info, Common",
  "User": "juan",
  "Message": "Usuario solicita lista de usuarios"
}
```

Componentes

Los distintos componentes

En esta sección se hará un análisis sobre el rol de cada uno de los componentes del sistema, al igual que los medios mediante los cuales se comunican.

Ciente del usuario

El cliente del usuario es una aplicación por consola, la cual está encargada de tomar los datos y archivos ingresados, para transferirlos al servidor principal con el fin de loguear, registrar y subir/comentar fotos. Este componente se comunica de forma exclusiva con el servidor mediante un protocolo definido que hace uso de TCP, con el fin de brindar una transferencia de información sin pérdida de bytes.

Server Instafoto

El servidor de Instafoto es un servidor gRPC el cual se encarga de almacenar toda la información de los usuarios, brindando servicios de alta/baja/modificación de usuarios así como la recepción de archivos y el registro de logs en otro servidor dedicado a dicha tarea.

Dicho servidor se comunica mediante TCP con los clientes de usuarios para enviar/recibir comandos y archivos. Por otro lado utiliza tecnología MOM (en este caso RabbitMQ) para transferir logs al server de logs y brinda un servicio gRPC a los clientes administrativos.

Server de logs

El server de logs se encarga de almacenar los logs generados por el server Instafoto, y es posible consumir dichos logs desde los clientes administrativos a través de nuestro protocolo TCP.

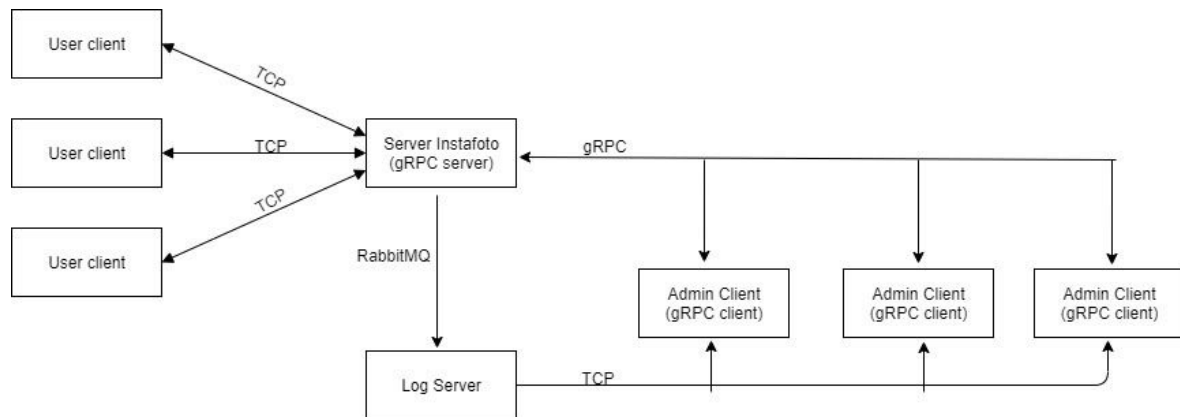
Ciente administrador

El cliente administrador se encarga de consumir el servicio gRPC que provee el server Instafoto al igual que consumir los logs almacenados en el server de logs mediante TCP.

Common

El componente common es utilizado por el resto de los componentes del proyecto. Este contiene las definiciones de los protocolos TCP (tanto las clases utilizadas como los métodos para su correcta función), como las clases de logs y sus métodos para serializarlos en formato Json.

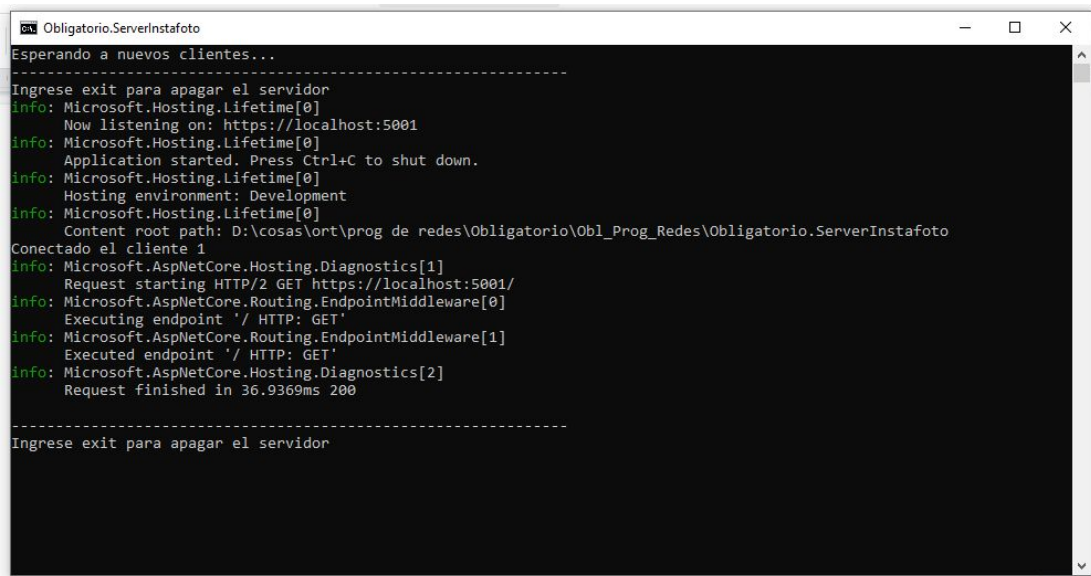
Esquema de los componentes



Uso de la aplicación

Para iniciar la aplicación se deben ejecutar todos los componentes (ctrl + F5 en visual studio), y se podrá iniciar nuevas instancias de los distintos clientes con el fin de tener nuevos usuarios.

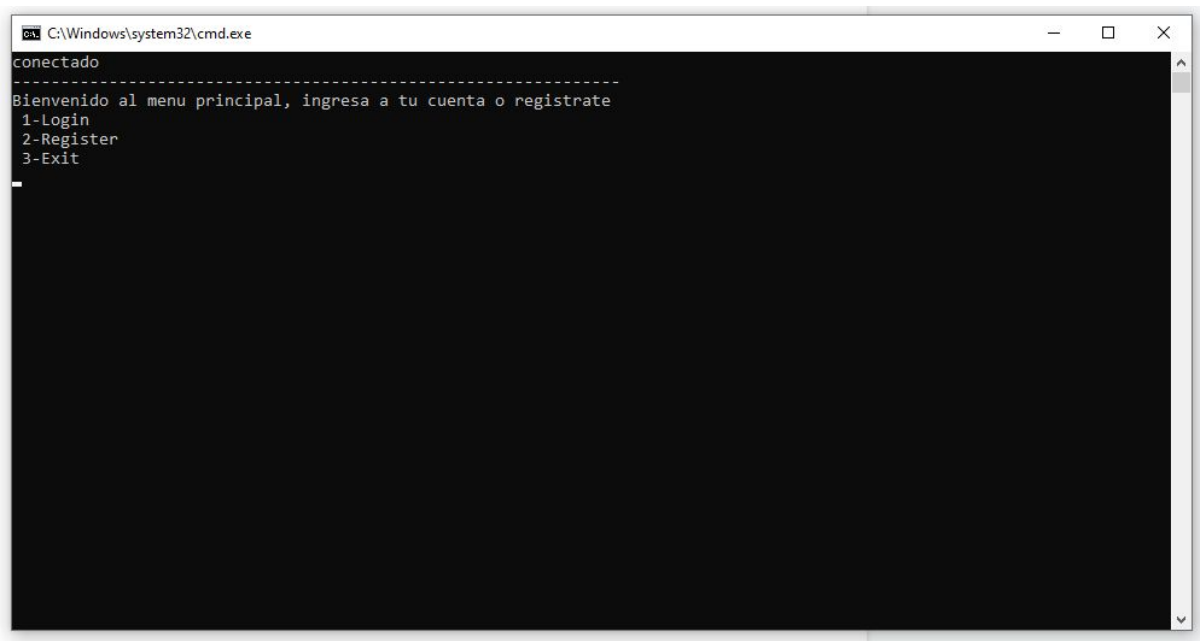
Server Instafoto



```
Obligatorio.ServerInstafoto
Esperando a nuevos clientes...
-----
Ingrese exit para apagar el servidor
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\cosas\ort\prog de redes\Obligatorio\Obl_Prog_Reses\Obligatorio.ServerInstafoto
Conectado el cliente 1
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 GET https://localhost:5001/
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[0]
      Executing endpoint '/' HTTP: GET
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[1]
      Executed endpoint '/' HTTP: GET
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished in 36.9369ms 200
-----
Ingrese exit para apagar el servidor
```

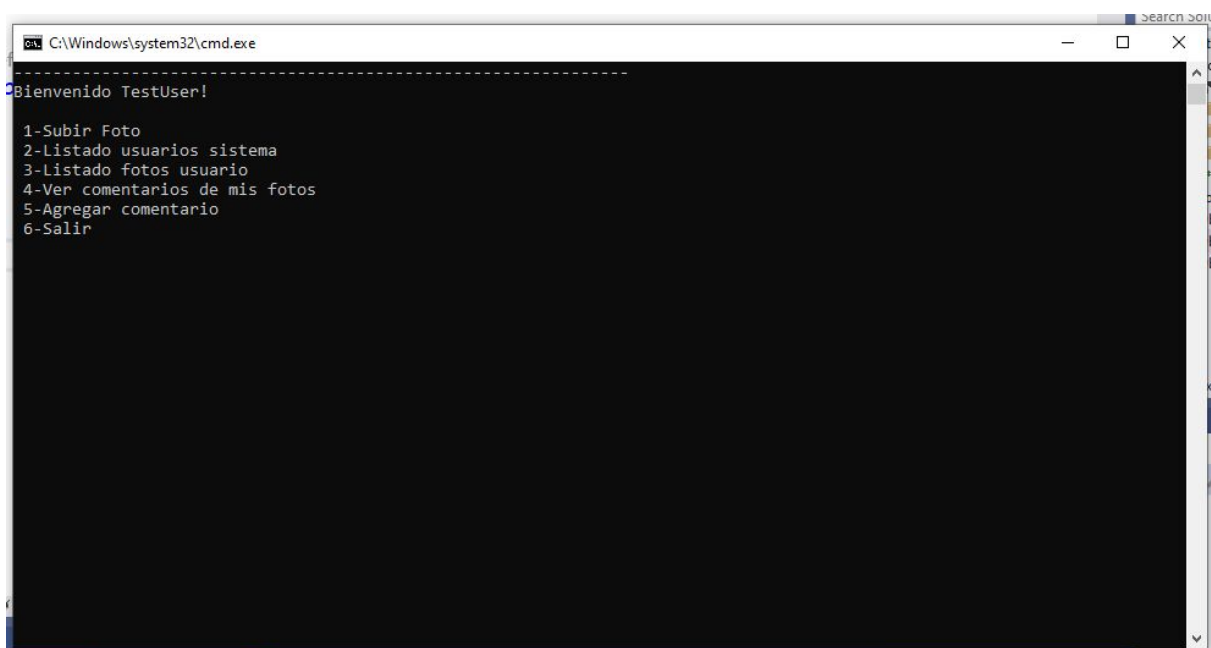
El server de instafoto consta de un único comando “exit”, el cual cierra el servidor junto a la conexión con todos los clientes conectados por TCP. También notifica cuando se conectan/desconectan distintos clientes.

Cliente usuario



```
C:\Windows\system32\cmd.exe
conectado
-----
Bienvenido al menu principal, ingresa a tu cuenta o registrate
1-Login
2-Register
3-Exit
```

Menú inicial anterior a un registro.



```
C:\Windows\system32\cmd.exe
-----
Bienvenido TestUser!
1-Subir Foto
2-Listado usuarios sistema
3-Listado fotos usuario
4-Ver comentarios de mis fotos
5-Agregar comentario
6-Salir
```

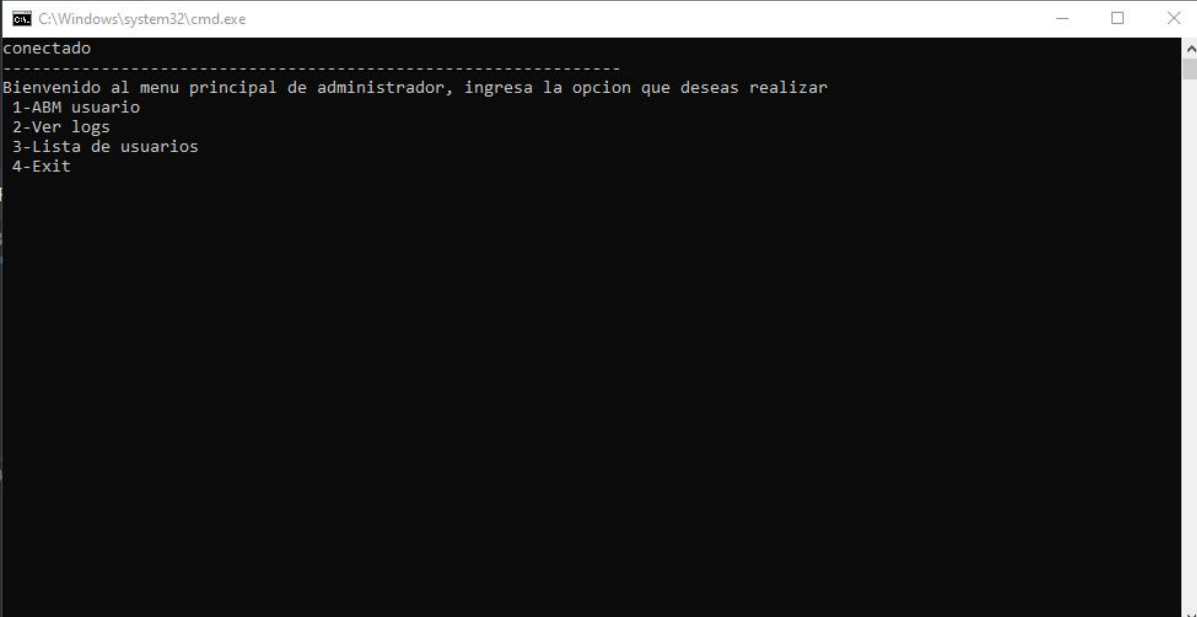
Menú principal una vez se registra/loguea un usuario.

Estos menús permiten al usuario crear una cuenta nueva o ingresar a una ya existente, así como realizar todas las acciones que permite el componente.

Log server

El servidor de logs funciona de forma pasiva, guardando los logs recibidos en una lista y devolviendolos a medida que se le solicitan nuevos.

Cliente administrador



```
C:\Windows\system32\cmd.exe
conectado
-----
Bienvenido al menu principal de administrador, ingresa la opcion que deseas realizar
1-ABM usuario
2-Ver logs
3-Lista de usuarios
4-Exit
```

El cliente administrativo expone una interfaz por consola que indica de forma explícita las distintas opciones que presenta, estas son: ABM usuarios, ver logs y listar los usuarios registrados en el sistema.

Link Repositorio

Link: <https://github.com/jcasanoval/ProgRedes.git>