PRedes - Obligatorio 2 - 254254 - Ismael Umpierrez

Proyecto git Público: https://github.com/I-Umpierrez-91/PRedes

Documentación

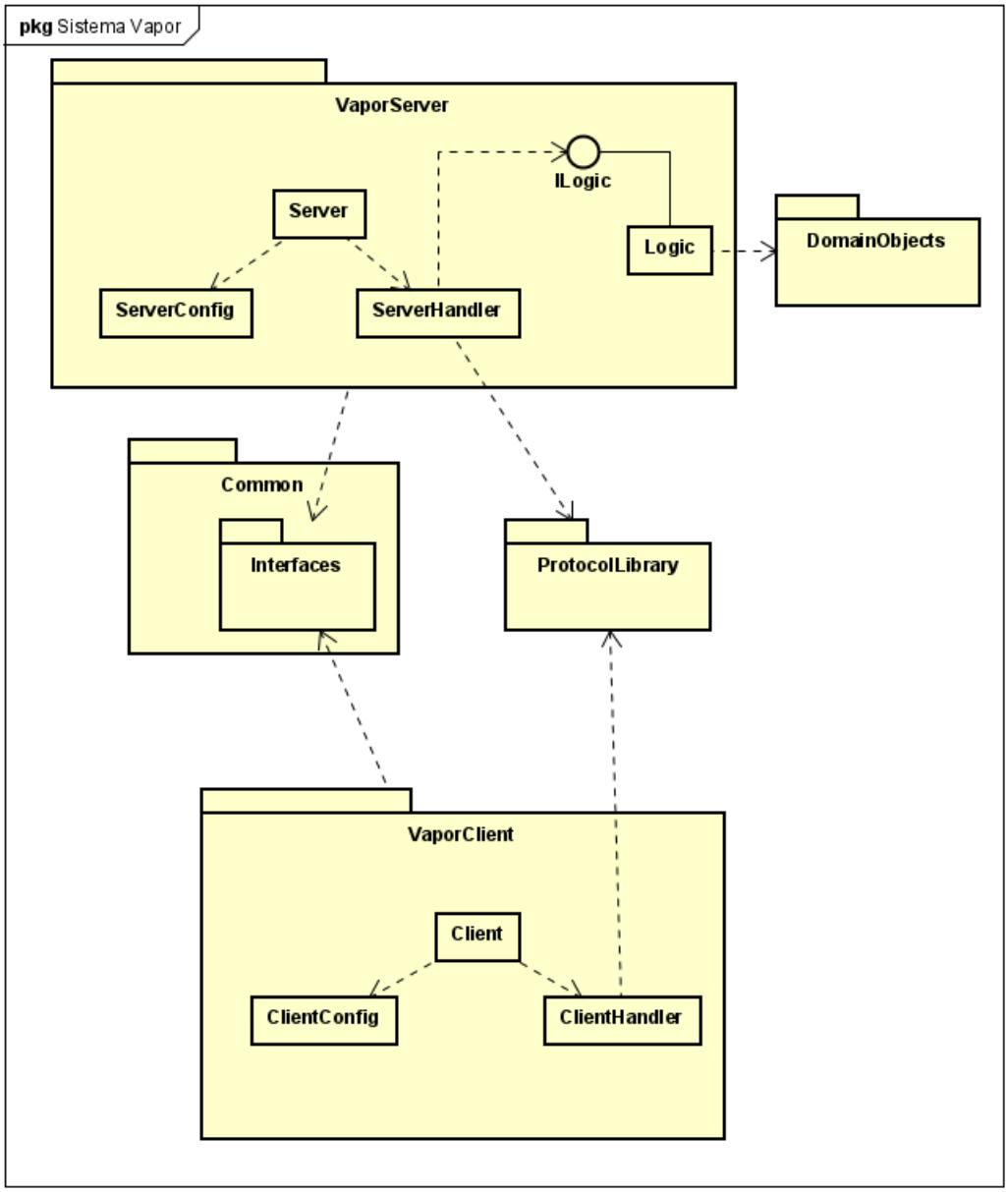
Diseño de la aplicación

Arquitectura de la aplicación

La aplicación consiste de dos módulos principales, cliente y servidor. Además se cuenta con proyectos auxiliares dentro de la solución que permiten desacoplar y reutilizar el código entre las dos aplicaciones principales.

Estas premisas se intentaron mantener en toda la extensión del obligatorio, se ha tratado de cuidar al máximo posible la legibilidad del código. También se ha hecho hincapié en la usabilidad de las aplicaciones resultantes.

Debajo se muestra un diagrama donde se puede ver la disposición general de los paquetes y para el caso de las aplicaciones principales la interacción entre sus componentes. También se intenta mostrar cuales son las dependencias.



Servidor

El servidor está contenido en el proyecto VaporServer. Dentro del Servidor se cuenta con una clase principal llamada Server que se encarga de manejar la interacción con el usuario administrador y de invocar a la clase ServerHandler que se encarga de manejar las requests de los clientes y mantener las conexiones con ellos.

La clase ServerHandler actua como controlador, y es la única que interactúa con la lógica, representada por la clase logic. Esta interacción se da a través de la inyección de dependencias usando la interfaz ILogic.

La clase Logic es la encargada de implementar la lógica de negocio y de mantener los repositorios de los datos, que son mantenidos en memoria con excepción de las imágenes que se guardan en el directorio donde corra la aplicación.

En el proyecto VaporServer también reside la carpeta test donde se incluye la clase TestData que se encarga de insertar datos de prueba y los archivos que se usan para las carátulas de los Juegos creados por esta clase.

Cliente

El cliente tiene una arquitectura interna parecida a la del servidor. Se encuentra en el proyecto VaporClient. Se cuenta con una clase principal llamada Client que se encarga de la interacción con el usuario y que también instancia y hace uso de una clase ClientHandler. Esta última tiene varias funcionalidades de apoyo a la primera, por ejemplo establece la conexión, contiene métodos para ayudar a escribir y leer mensajes a través del protocolo utilizado y contiene parte de la inteligencia usada para recibir archivos.

Clases auxiliares

Como parte de los esfuerzos en pos de la reutilización del código, pero también para ayudar a desacoplar y entender mejor el mismo, se cuenta con una serie de proyectos/clases auxiliares donde se derivan algunas tareas repetitivas.

En el proyecto Common se tienen handlers que se encargan de manejar el NetworkStream (NetworkStreamHandler), el FileStream (FileStreamHandler) y el manejo de archivos en general (FileHandler). Como punto a destacar, el uso de estas clases siempre se hace inyectando las dependencias a través de las interfaces que se encuentran en el mismo proyecto.

El proyecto DomainObjects simplemente contiene la definición de las clases base, de manera de aportar más claridad y más orden a la codificación del proyecto. Por el momento no hay lógica de manejo de base de datos, pero de implementarse la incluiría ahi.

Por último se tiene el proyecto Protocol Library que se encarga de definir el protocolo y algunas de sus funcionalidades más utlizadas. Se ahondará más sobre el protocolo en la siguiente sección.

Protocolo utilizado

El protocolo elegido fue el que se sugirió en la letra del obligatorio con una pequeña diferencia, la diferencia consiste en que se agrega un sector de datos opcional luego del header que contiene los bytes correspondientes al archivo si es que se quiere enviar uno.

Otra decisión importante respecto al protocolo es que cuando se quieren enviar conjuntos de datos independientes, estos se separan en el mensaje usando un divisor que está definido en la clase HeaderConstants.

Protocolo:

Nombre del campo	Header	CMD	Largo	Datos	Datos adicionales (Opcional)
Valores	RES/REQ	0-99	Entero	Variable	Variable
Largo	3	2	4	Variable	Variable

El proyecto ProtocolLibrary contiene todo lo relativo al protocolo, la clase CommandConstants contiene la lista de los comandos permitidos incluyendo los ids de cada una de las operaciones. Ambas aplicaciones interactúan sacando las constantes de ahi.

La clase Header contiene funciones útiles para armar y desarmar los mensajes. Se provee también la funcionalidad de armar templates de headers para respuestas y requests.

Manejo del paralelismo

Para manejar el paralelismo se usa la clase Task. Esto comprende un cambio respecto al obligatorio 1 donde usabamos Threads. El paralelismo se aplica cuando el servidor recibe la solicitud de conexión de un cliente, al establecer la conexión se inicia un Task correspondiente al nuevo cliente.

Además se hizo uso del patrón async await en otras partes del código, por ejemplo en los handlers de las aplicaciones.

Manejo de la mutua exclusión

Para asegurar la mutua exclusión se hizo uso de Lock. Cada vez que se quiere modificar una de las colecciones del sistema la misma se bloquea para asegurar que no hay lecturas sucias. Esto también es un cambio respecto al obligatorio 1 donde no había podido ser aplicado.

Limitaciones y errores conocidos

Se sabe que tanto la aplicación cliente como la aplicación servidor no manejan de la manera más agraciada los casos de uso donde se parte de la lista de juegos y se sigue con alguna acción cuando no hay ningún juego cargaco.

Un ejemplo de esto es cuando se quiere comprar un juego y todavía no hay ninguno, se solicita igualmente el número de juego.

Los datos sensibles como el password no se manejan de ninguna manera particular respecto de los datos comunes.

La aplicación no siempre se comporta bien ante cortes abruptos en la comunicación.

Funcionalidades implementadas

Para este obligatorio se implementaron todas las funcionalidades solicitadas, agregando las que habían faltado en la entrega anterior y trabajando para mejorar las existentes.

1. Conexión y desconexión de un cliente al servidor.
2. Alta, baja y modificación de usuarios (sólo servidor).
3. Login y logout al sistema.
4. Ver catálogo de juegos disponibles.
5. Que un usuario adquiera un juego.
6. Ver los juegos que adquiridos por el usuario.

7. Publicar un juego.
8. Publicar la carátula del juego.
9. Descargar una carátula de un juego.
10. Publicar una calificación del juego (luego del promedio se va obtener la nota del juego).
11. Publicar una reseña escrita de un juego.
12. Buscar Juegos (por los diferentes filtros ya definidos).

Información para el test

Al compilar la aplicación se crean los ejecutables para el cliente y el servidor. Están en \VaporClient\bin\Debug\net5.0 y \VaporServer\bin\Debug\net5.0 respectivamente.

Se provee la funcionalidad de carga de datos de prueba en la aplicación servidor. Para ejecutarla solamente tienen que seleccionar la opción 99 en el menú. A continuación se provee un resumen de los datos de prueba:

Usuarios:

UserName	Password
Jorge	Jorge
Maria	Maria

Juegos:

Nombre	Género	Carátula
Paper Mario	Arcade	PaperMario.jpeg
Microsoft Flight Simulator	Simulador	FlightSimulator.jpeg
Wasteland 3	RPG	Wasteland3.jpeg

Juegos de Jorge:

Nombre
Paper Mario
Microsoft Flight Simulator

Juegos de María:

Nombre
Microsoft Flight Simulator
Wasteland 3

Reviews:

Usuario	Juego	Nota	Titulo
Jorge	Paper Mario	3	Ta bueno
Jorge	Microsoft Flight Simulator	5	Me encanta este juego
Maria	Microsoft Flight Simulator	4	Esta demas, falta un poco de graficos
Maria	Wasteland 3	1	No me gustó, un bajon