Primer Obligatorio

Programación de Redes

Grupo N6A

2019

Docente:
Roberto Martín Assandri

Integrantes:
Rodrigo Bertolotti 220818
Rodrigo Duarte 217374

Introducción

El objetivo del proyecto es simular una plataforma similar a la utilizada en la facultad, donde haya cursos disponibles para que los alumnos se inscriban, se puedan subir materiales y los profesores puedan corregirlos, notificando a los alumnos con su calificación.

La comunicación entre las partes se realiza mediante un cliente y un servidor, los cuales se envían mensajes por medio de un protocolo programado por nosotros mismos. Múltiples clientes deben poder conectarse al servidor, quien es el encargado de recibir las peticiones, interpretarlas, procesarlas y responderlas.

Aplicación cliente

RF1: Conectarse al servidor.

Se deben poder conectar múltiples usuarios a la vez al servidor.

RF 2: Login de usuario.

El usuario debe poder acceder al sistema ingresando su número de estudiante o email y contraseña.

RF 3:Registro a curso.

Descripción: El sistema debe permitir dar de alta un alumno a un curso. Se deben listar los cursos disponibles para inscripción y que el usuario decía a cual anotarse.

RF4: Consultar cursos disponibles.

El sistema debe listar los cursos disponibles para que el usuario se inscriba a los mismos.

RF 5: Notificar alumno.

Una vez que una nota es asignada a un alumno, el sistema debe notificar a dicho alumno que la nota fue asignada, especificando la nota y el material que fue evaluado.

RF6: Consultar notas recibidas.

El alumno debe poder consultar todas las notas que recibió previamente.

Aplicación servidor

RF1: Aceptar pedido de conexión de clientes

La aplicación servidor debe poder aceptar pedidos de conexión de múltiples clientes a la vez.

RF 2: Dar de alta a un alumno

La aplicación servidor debe poder crear alumnos y darlos de alta. Solamente luego que la aplicación servidor realiza dicha tarea el alumno puede realizar el login del lado del cliente.

RF3: Dar de alta un curso.

La aplicación servidor debe poder crear cursos y darlo de alta. Una vez que el curso es creado, esta estará disponible para que los alumnos se insciban del lado del cliente.

RF4: Dar de baja un curso.

Se debe poder borrar cursos. Una vez que el xcurso es borrado este no estará mas disponible para ser accedido por los alumnos del lado del cliente.

RF5: Dar de alta un alumno a un curso.

Se debe poder inscribir un alumno a un curso del lado del servidor. Una vez que esto sucede, el alumno podrá subir materiales a dicho curso del lado del cliente.

RF6: Subir material a curso.

Un usuario que está inscripto a un curso debe poder subir un material a dicho curso.

Rf7: Listar cursos

Se deben poder listar todos los cursos disponibles para que los alumnos se inscriban.

RF 6: Asignar nota a alumno.

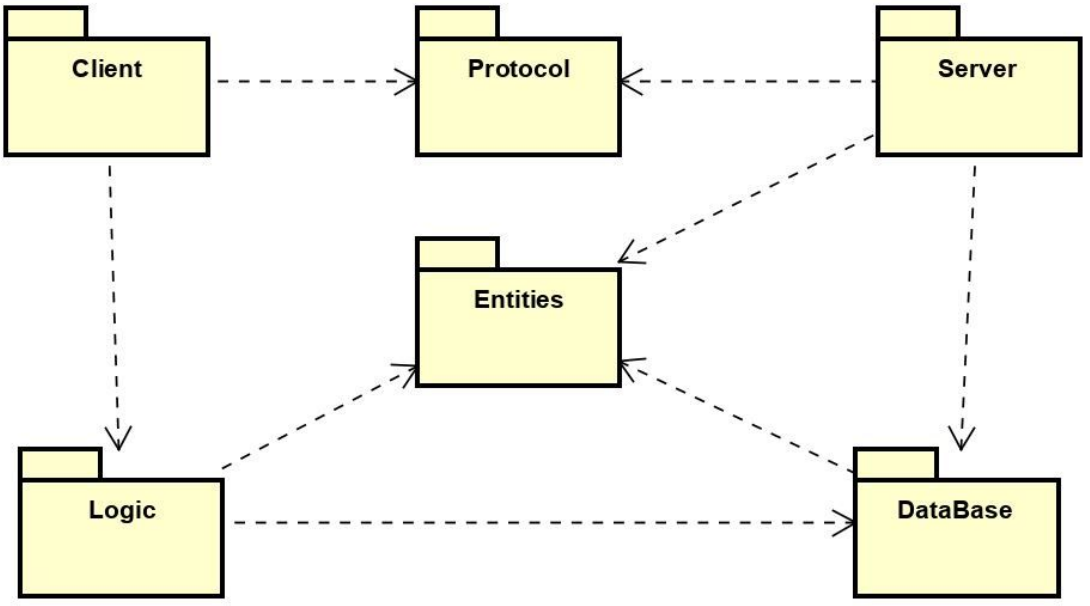
El administrador del sistema debe poder asignar una nota a un alumno que haya subido un material a un curso determinado.

RF 7: Notificar alumno.

Una vez que una nota es asignada a un alumno, el sistema debe notificar a dicho alumno que la nota fue asignada, especificando la nota y el material que fue evaluado.

Estructura del proyecto

Dividimos el proyecto en 6 paquetes diferentes, cada uno tiene una responsabilidad bien definida y sus funcionalidades son implementadas en las clases que son detalladas más adelante.

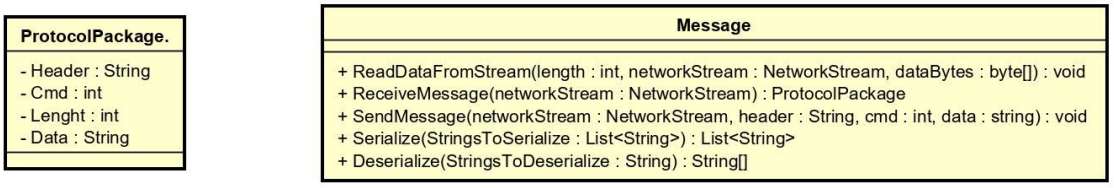


Package Protocol:

La comunicación entre el cliente y el servidor se realiza mediante un protocolo realizado por nosotros, el cual está implementado en el package protocol.

Los datos son enviados por el método SendMessage y recibidos por el método RecieveMessage. Los mensajes almacenados en el buffer son leídos por medio del método ReadDataFromStream.

Los mensajes enviados del cliente al servidor y viceversa se mandan en contenidos en un paquete cuya estructura está definida en el método ProtocolPackage.



Header:

Indica el tipo del paquete, siendo este REQ (request) o RES (response).

CMD:

El comando que indica que tipo de request el cliente le está realizando al servidor. Un switch contenido en la clase Program del Server se encarga de invocar a los métodos responsables de responder a cada petición.

Length:

Largo del paquete que se desea enviar.

Data:

Información contenida dentro del paquete. Para el envío e interpretación de los mensajes enviados utilizamos los métodos Serialize() el cual se encarga de separar las diferentes partes de los datos que queremos enviar por medio de el carácter “;” y Deserialize() que devuelve un array de strings, en el cual dentro de cada índice del array se encuentra una parte distinta de los datos que se enviaron.

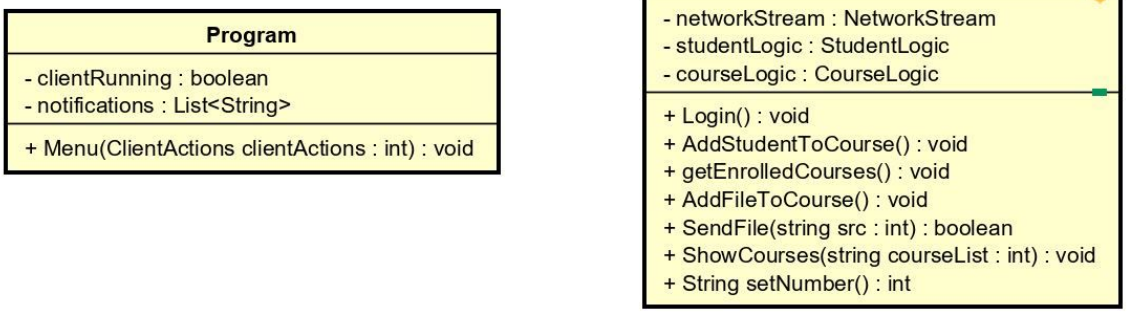
Esta estructura nos permite agregar nuevas funcionalidades sin mayor impacto en el código, que solamente debemos crear nuevos comandos que se correspondan con las nuevas funcionalidades que se desea implementar.

Package Client:

Es el paquete que se encarga de definir el comportamiento del cliente. En la clase Program se realiza la conexión al servidor y en la clase ClientActions se realizan las request al servidor y los métodos asociados.

Se crea un nuevo TcpClient por cada nuevo cliente que se desea conectar, por medio del cual se enviaran los request al server.

Por otra parte, se crea un TcpClient llamado tcpClientBackground que se encarga de mantener en espera al cliente para recibir notificaciones del servidor para los casos en donde se asigna una nota a un curso.

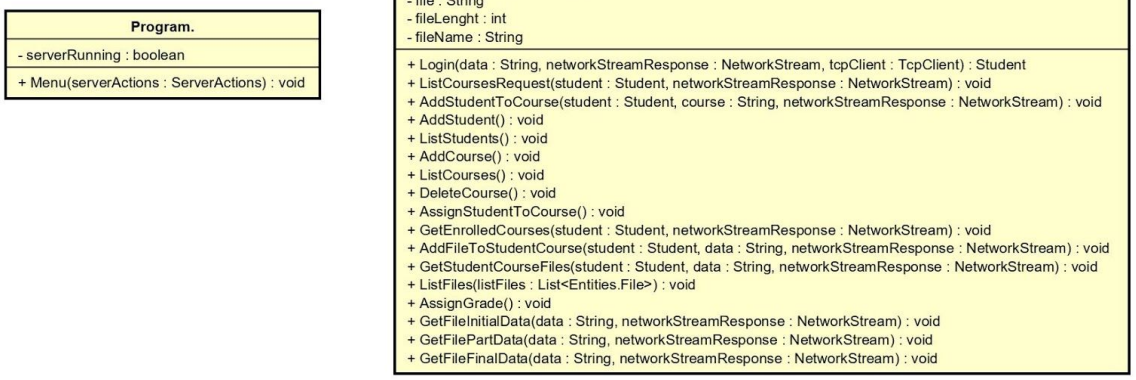


Package Server:

En la clase Program del proyecto Server es donde se aceptan las conexiones de los clientes hacia el servidor y se traducen los comandos indicados en las peticiones que llegan del lado del cliente. En la clase ServerActions se hace el procesamiento de los request realizados por el cliente y se hacen los response correspondientes en cada caso.

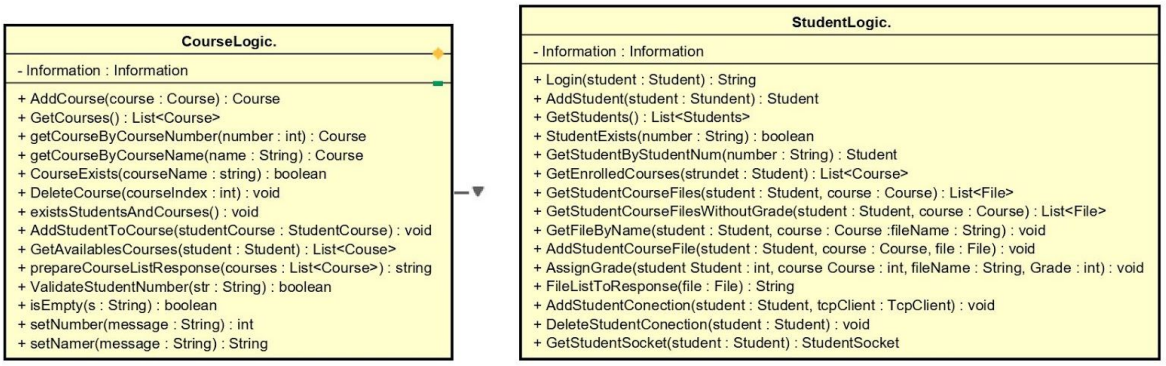
Se crea un tcpListener que se mantiene a la espera de nuevos clientes siempre y cuando el servidor siga corriendo. Cada vez que se conecta un nuevo cliente, se abre un nuevo thread el cual se encarga de interpretar los comandos que llegan en los paquetes enviados desde el cliente. Una vez que llega un paquete, el método switch dentro del Thread recibe el comando y llama al método encargado de procesar los request.

Dichos métodos se encuentran en la clase serverActions, y son los encargados de realizar las respuestas a los response.



Package Logic:

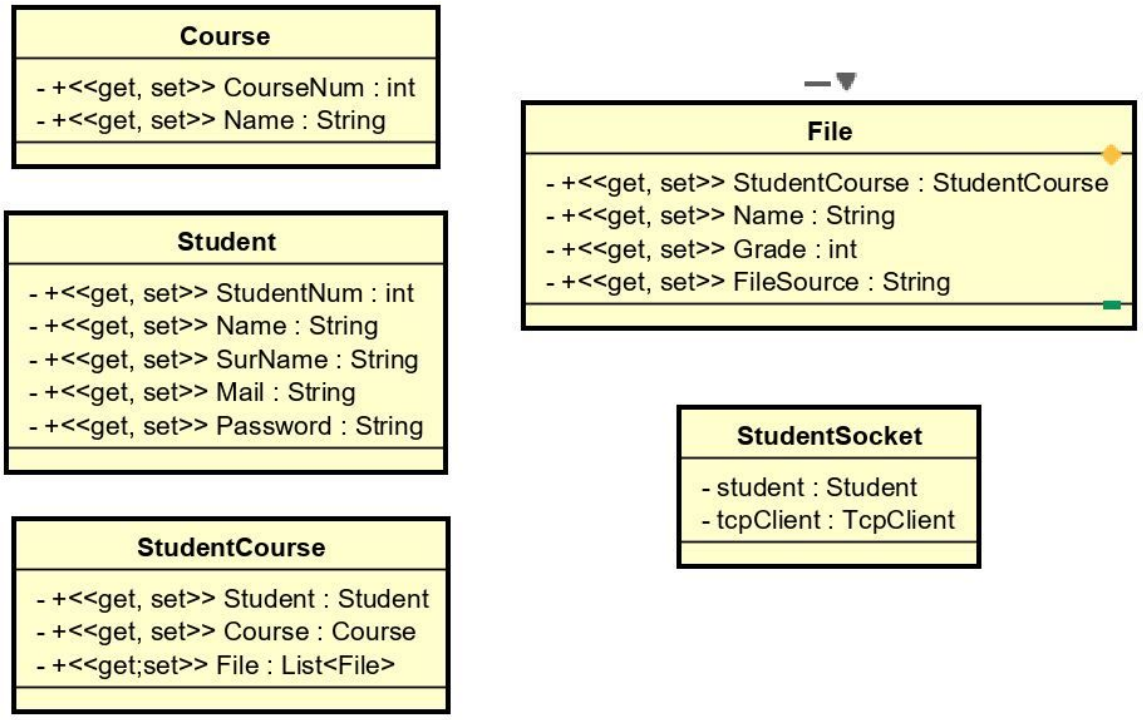
En este paquete se encuentra la lógica que involucra a los estudiantes y los cursos en las clases StudetLogic y CourseLogic respectivamente. Contiene los métodos utilizados para realizar las validaciones de datos pertinentes en cada caso, y en el caso de StudentLogic, contiene los métodos que se encargan de permitir que un alumno suba archivos a un curso.



Package Entities:

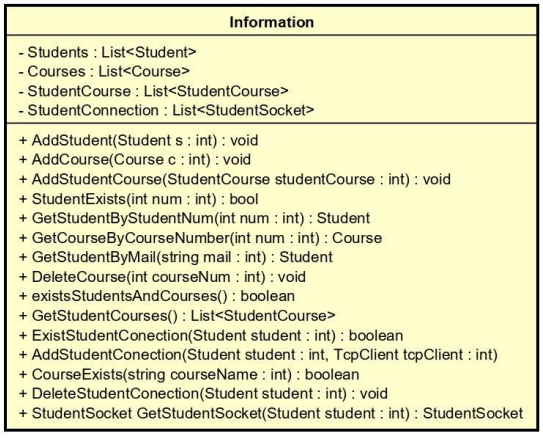
En este paquete se crean y se definen las entidades que componen a la aplicación, cada una con sus atributos y properties. Creamos una clase para Cursos y para estudiantes con sus atributos. También tenemos una clase llamada StudentCourse, la cual representa a los estudiantes inscriptos a una materia determinada, el atributo file es una lista de archivos que el estudiante subió a ese curso para esa materia, los cuales podrán ser calificados.

Por otra parte, creamos una entidad StudentSocket que relaciona a un estudiante con su Socket. La utilizamos para conocer los usuarios conectados para realizar la comunicación en dirección Servidor-Cliente.



Package DataBase:

Aquí se definen las listas las cuales manejara el servidor. Cabe aclarar que este obligatorio persiste los objetos en memoria, y debido a la arquitectura empleada, el encargado de utilizar y modificar las listas de la base de datos es siempre el servidor. Los clientes no tienen acceso a los datos de la base de datos. La clase Information contiene todas las listas y los métodos encargado de realizar distintos tipos de operaciones sobre ellas.



Subida de archivos

Primero que nada, para el manejo de archivos creamos una clase File en el package Entities, la cual es la encargada de crear objetos que tengan todos los atributos que los archivos requieren.

El envío de archivo se divide en tres etapas:

Etapa inicial: En la etapa inicial se envían los datos iniciales al servidor y se confirma el comienzo de envio del archivo. Estos datos incluyen nombre del archivo y tipo de extensión.
Etapa media: En esta etapa se define un tamaño fijo de 999 bytes. Luego se envía la información del archivo en partes de ese tamaño hasta completar toda la información.
Etapa final: Cuando se detecta que se envió hasta la última parte del archivo, esta etapa se encarga de ejecutar el último método el cual persiste el archivo en una carpeta determinada.

Del lado del servidor los encargados de llevar a cabo esas tres etapas son los siguientes métodos: GetFileInitialData, GetFilePartData y GetFileFinalData.

Estudiantes conectados

El sistema cuenta con una colección en la clase Informacion del DataBase encargada del manejo de estudiantes conectados al sistema.
Cada vez que un estudiante se loguea al sistema este se considera como un estudiante conectado, si el mismo desea conectarse desde otra terminal se le muestra un mensaje de error informando que ya se encuentra conectado.
Una vez que el estudiante sale del sistema o la terminal es cerrada, este estudiante se borra de la colección de estudiantes conectados y puede volver a conectarse desde otra terminal.
Por otro lado, en esta misma colección se guarda el Socket secundario de cada cliente, el cual es usado cada vez que se le desea enviar una notificación, lo cual se explica a continuación.

Envío de notificaciones

Como se dice en secciones anteriores, para el envío de notificaciones se crea un Socket secundario exclusivamente para esta funcionalidad, este es el Socket guardado en la colección de estudiantes conectados. Con este, cada vez que del lado del servidor se asigna una nota, una notificación es enviada al cliente.
Del lado del cliente, se cuenta con un Thread desde el cual se está conectado al Socket, en el que continuamente se está esperando recibir una notificación.
Cada notificación se guarda en una colección de notificaciones que tiene cada cliente y en el momento adecuado se muestran todas las notificaciones en pantalla.
Una vez impresas las notificaciones la colección es reiniciada.
En el caso que el cliente no se encuentre conectado, la notificación no es enviada.

Manejo de concurrencia

Utilizamos locks para bloquear el acceso a las listas que pueden ser modificadas y accedidas por el servidor y el cliente respectivamente, eliminando el problema de la mutua exclusión. Por ejemplo, creamos un lock para bloquear la lista de cursos, en el caso que un curso sea borrado del lado del servidor, este no podrá ser accedido del lado del cliente para que un usuario se inscriba a él mismo.