Universidad ORT Uruguay

Facultad de Ingeniería

Programación de Redes
Obligatorio II

Juan Pablo Carbonell - 203357
Mathias Lantean - 180094
Docente: Roberto Martín Assandri

Entregado como requisito de la materia Programación de Redes

Miércoles, 20 de Noviembre de 2019

Declaraciones de autoría

Nosotros, Mathias Lantean y Juan Pablo Carbonell, 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
- 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.

Dedicatoria y agradecimientos

Agradecemos al docente por darnos una corrección objetiva y comentarios sobre que corregir del obligatorio anteriormente entregado. Así como también, por su consideración sobre este obligatorio teniendo en cuenta que cursabamos otras materias que consumían muchisimo tiempo (Diseño de aplicaciones 2).

Otro gran agradecimiento a todos los compañeros del curso quienes hicieron consultas en clase, resolviendo dudas que nosotros también teníamos.

Y por último a la cantina y biblioteca de ORT que nos permitió juntarnos por largas horas en sus instalaciones.

Resumen

Es la continuación de la tarea presentada para la primer instancia. El obligatorio consiste en la implementación de una aplicación cliennte/servidor que permita gestionar estudiantes, cursos y realizar entregas de tareas.

Para esta oportunidad se extiende a nuevos conceptos vistos en clase tales como manejo de tasks, MOM, RPC/ORB y manejo de web services.

Ambos aplicaciones son de consola dado que el foco del obligatorio esta en la interconexión entre las mismas y además se proporciona una API rest la cual puede ser consumida por cualquier herramienta de peticiones de APIs (Ej. postman).

En este documento se detalla las decisiones de diseño, componentes y arquitectura del sistema.

Palabras claves

- Socket: Es un endpoint interno para enviar o recibir datos dentro de un nodo en una red.
- Dominio: Cuando hablamos del dominio o dominio del problema nos referimos al alcance del proyecto, es la situación o problema que el sistema a implementar debe ser capaz de resolver.
- Thred: Es la secuencia más pequeña de instrucciones programadas que un sheduler puede administrar de forma independiente, que generalmente forma parte del sistema operativo.
- Multithreading:ees la habilidad de la CPU para proveer múltiples threds de ejecución de forma concurrente.
- TCP: Por sus siglas en inglés de Transmission Control Protocol es un estándar que define cómo establecer y mantener una conexión de red a través de la cual se pueden intercambiar datos. TCP funciona con el Protocolo de Internet (IP), que define cómo las computadoras envían paquetes de datos entre sí.
- Task: hace referencia a async task, una tecnología de C# que proporciona una abstracción sobre código asincrónico con la idea de que se leea como una secuencia de declaraciones, pero se ejecuta en un orden mucho más complicado basado en la asignación de recursos externos y cuando se completan las tareas.
- MSMQ: es el gestor de colas de Microsoft. Se utiliza normalmente para enviar mensajes u objetos entre aplicaciones de una manera disociada.
- API: Es un conjunto de funciones y procedimientos que permiten la creación de aplicaciones que acceden a las características o datos de un sistema operativo, aplicación u otro servicio.

Índice general

1. Alcance del proyecto

Se debe diseñar sobre el sistema ya existente una solución que permita ver un log de todos los eventos del sistema, se pretende también permitir que se den de alta docentes y que los mismos corrijan materiales.

Se define el acceso a nuevas funcionalidades mediante HTTP y dado que por requerimiento las funcionalidades del antiguo servidor no se pueden exponer mediante esta tecnología, se realiza un pasamano entre tecnologías para lograr el objetivo.

2. Protocolo

El protocolo consiste en el envío de tres paquetes desde el cliente al servidor, conformando una trama. Los 2 primeros son numéricos y especificarán por un lado la acción que el servidor debe realizar y por otro el largo del payload en bytes y por otra parte los datos propiamente dichos en bytes.

Nota: En un principio se había incluido un paquete extra el cual indicaba el tipo de dato que se enviaba dentro del payload con la finalidad de parsearlo correctamente. Luego determinamos que ya sabemos lo que debería venir en el payload de acuerdo a la acción que se envió.

`action:` funcionará como lo hace una URL en una Request HTTP. Indicará el recurso a consumir. Por ejemplo Login, GetUserList, SendTask, etc. Las mismas estarán definidos en un enumerado `(ya que facilita el envío y recepción al ser un dato numérico)` y dependiendo el elegido será la acción que el servidor realizará.

`data_length:` es el largo del resultado de transformar el string a enviar en array de bytes

`data:` array de bytes del string que se enviará. El contenido del string enviado es parseado separando los atributos por '&' y en caso de ser una lista de elementos, los mismos se separan por ';'. Este parseo de la información es bastante sencillo y a pesar de que es necesario considerar el orden posicional de los atributos en el string, nos permite mandar la información de forma estructurada.

3. Funcionamiento

Actualmente la aplicación cuenta de 3 proyectos principales independientes los cuales se relacionan entre si. La aplicación cliente permanece intacta a la entrega anterior. La aplicación servidor ahora es más robusta y tiene más responsabilidades. Por otra parte, con el fin de satisfacer los nuevos requerimientos, se creo un nuevo proyecto llamado `WebApi` el cual será encargado de exponer una interfaz API REST.

3.1. Servidor

El proyecto del servidor mantiene las dos responsabilidades principales de la entrega anterior. Por un lado es el encargado de la gestión de estudiantes y operaciones que los administradores pueden realizar y por otra parte es el encargado de recibir y responder peticiones de la aplicación cliente. Cuando el servidor se inicia se crea un hilo para cada una de estas dos responsabilidades.

Además, este servidor, es el servidor de remoting

3.2. Consola administrador y estudiante

Las funcionalidades de estas consolas no variaron de la implementación realizada para el obligatorio anterior.

Existe un único administrador y sólo existirá una única instancia del servidor en ejecución, las operaciones que el administrador realiza son directamente ejecutadas en la aplicación servidor, no hay necesidad de transferir datos, ya que el servidor es el encargado de gestionar el repositorio de datos.

En cuanto a las operaciones que el estudiante realiza si existe una transferencia de datos utilizando el protocolo elegido. (Su funcionamiento es detallado con más precisión en la documentación del obligatorio I)

3.3. Resolución de peticiones

El servidor es el encargado de escuchar las peticiones de las aplicaciones clientes `(pudiendo existir muchas instancias en ejecución de estas últimas)`. Las peticiones son realizadas utilizando el protocolo descrito anteriormente, de modo que se creó una función "dispatcher", esta función es la encargada de ejecutar dinámicamente la función adecuada para procesar la petición solicitada con la información recibida.

3.4. Manejo de errores

En cuanto al código refiere, los errores se controlan mediante excepciones que son capturadas en los correspondientes hilos. En las ocasiones que no se desea lanzar una excepción se realizan verificaciones de los objetos a utilizar, correctitud de los datos, etc.

Si nos referimos al manejo de errores entre el cliente y el servidor, lo que hacemos es por parte del cliente enviar la solicitud y esperar por la respuesta del servidor la cual será ejecutada siguiendo el protocolo definido para mantener la homogeneidad del envío de datos entre ambas aplicaciones.

3.5. Control de concurrencia

Para el manejo de la concurrencia se creó una clase singleton llamada "DataSystem"la cual es la responsable de mantener la listas de los objetos de nuestro dominio, básicamente es el repositorio de los datos.

Esta clase al ser singleton debe ser cuidadosamente manejada dado que puede existir conflictos en caso de que se solicite la instancia desde dos sitios al mismo tiempo. Para evitar este problema, al momento de obtener la instancia o crearla, se establece un semáforo y el constructor de la clase es privado, de modo que la única forma de obtener una instancia es pasando por el semáforo.

"DataSystem"también brinda las operaciones de ABM de los objetos del sistema. Como estas operaciones son altamente concurrentes, se crea un semáforo para cada una de las listas del sistema y se utilizan al momento de realizar la operación en una zona crítica.

Cabe destacar que anteriormente se utilizaban locks en lugar de semáforos, pero de la forma en la que se inicializan los semáforos, ambos funcionan exactamente de la misma manera.

4. Arquitectura y componentes

4.1. Diagrama general del proyecto

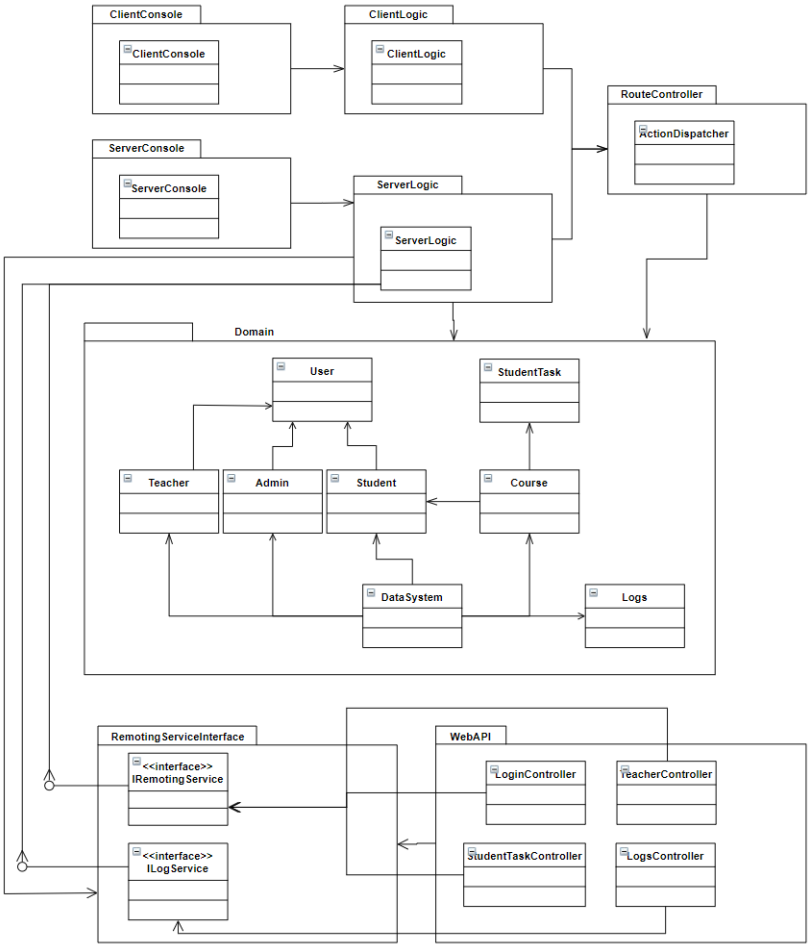

Figura 4.1: Diagrama general del proyecto

Para ver el diagrama mas detallado click aquí

4.2. Arquitectura

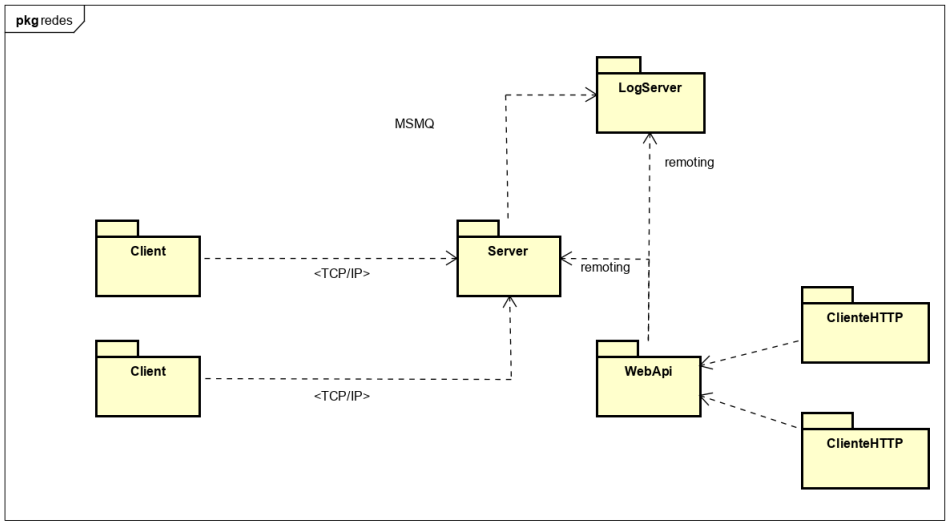

Figura 4.2: Diagrama de conexión - Arquitectura

4.3. Descripción de proyectos

ClientConsole

Este proyecto está constituído por una única clase "ClientConsole" que se comporta como una interfaz entre el cliente real (obteniendo datos), y nuestra clase cliente que utiliza esos datos para enviar solicitudes al servidor.

ClientLogic

Este proyecto está constituído por una única clase "Client" recibe datos desde la consola del cliente, y utiliza esos datos para enviar solicitudes al servidor, es quien se encarga de armar las tramas que se enviarán a través del socket al servidor.

ServerConsole

Este proyecto está constituído por una única clase "ServerConsole" que es utilizada principalmente por el administrador del sistema, pues aquí es donde se encuentra la interfaz que le permite realizar la ABM de estudiantes, cursos y tareas, así como realizar correcciones de tareas, etc.

ServerLogic

Este proyecto está constituído por una única clase "Server" que contiene la lógica que le permite efectivamente llevar a cabo las tareas solicitadas por el administrador desde la consola del servidor.

RouteController

Este proyecto está constituído por una única clase "ActionDispatcher" que se encarga de recibir las solicitudes del cliente y de acuerdo a la acción a realizar, llama a la función encargada de realizar dicha acción, es también quien conoce y modifica el Dominio del sistema.

Domain

Este proyecto está constituído por una clase repositorio "DataSystem" donde se encuentran todas las listas, necesarias del sistema y sus locks, y es quien gestiona el acceso a las mismas. Además el dominio contiene 4 clases que sirven para representar el negocio: Admin para representar al administrador, Student para representar a los estudiantes, Task para representa a las tareas y Course para representar a los cursos, donde se encuentran los estudiantes que cursan ese curso y las tareas relacionadas al curso.

WebApi

Este proyecto se encarga de exponer los endpoints mediante REST. La documentación de las APIs es detallada más adelante. Básicamnte es el encargado de exponer los servicios de los docentes.

RemotingServiceInterface

Define las interfaces necesarias para poder realizar la conexión remoting. Cabe destacar que estas interfaces las utiliza tanto el servidor como el cliente.

5. Ejecución del proyecto

En cada uno de los compilados podemos encontrar un archivo de configuración llamado “serverconsole.exe.config” y “clientconsole.exe.config” en donde debe especificarse la IP del servidor, el puerto del servidor, la IP del cliente y el puerto del cliente a utilizar.

Nota: Estos archivos de configuración fueron creados con el fin de evitar recompilar el código cada vez que se desea modificar alguna IP o puerto tanto de la aplicación cliente como de la aplicación del servidor.

Para el funcionamiento de la aplicación es necesario ejecutar en primer lugar la aplicación del servidor para luego sí ejecutar las diferentes instancias de la aplicación cliente.

5.1. Procedimiento de ejecución

1. Levantar el servidor `(Server)`.
2. Levantar el cliente antiguo para acceder a las funcionalidades de los estudiantes.
3. Levantar el cliente HTTP para acceder a las funcionalidades de los docentes y observar el log de los eventos. `(WebAPI)`.
4. Los últimos 2 pasos son opcionales, dependiendo de las funcionalidades que quieras utilizar, abrirás uno u el otro, o ambos.

6. Descripción de Endpoints

En nuestro proyecto existen 4 controladores, cada uno con sus funcionalidades específicas. Se adjunta a esta documentación una collection de postman la cual contiene el esqueleto para testear cada una de las APIs.

6.1. Teacher Controller

Aquí se detallan las funcionalidades de crear/obtener docentes.

6.1.1. Crear docente

URI	api/teacher
Verbo	POST
Headers	Content-Type: application/json

Body	{
	“Name”: “Nombre”,
	“Surname”: “Apellido”,
	“User”: { “Email”: “Email”,
	“Password”: “Contraseña”}
	}

Respuesta	{
	“Name”: “Nombre”,
	“Surname”: “Apellido”,
	“User”: { “Email”: “Email”,
	“Password”: “Contraseña”}
	}

6.1.2. Obtener docente

URI	api/teacher/{email}
Verbo	GET
Headers	Content-Type: application/json
Body	

Respuesta	{
	“Name”: “Nombre”,
	“Surname”: “Apellido”,
	“User”: { “Email”: “Email”,
	“Password”: “Contraseña”}
	}

6.2. Login Controller

Aquí se detalla la funcionalidad de iniciar sesión.

6.2.1. Login de docente

URI	api/login
Verbo	POST
Headers	Content-Type: application/json

Body	{
	“Email”: “Email”,
	“Password”: “Contraseña”
	}

Respuesta	token : Guid

6.3. Logs Controller

Aquí se detalla la funcionalidades relacionadas con obtener los logs y filtrar los logs.

6.3.1. Obtener logs

URI	api/logs
Verbo	GET
Headers	Content-Type: application/json
Body	
Respuesta	[“Fecha: 25/11/2019 20:24:27 -> Descripción: Se ha creado el curso: curso”, “Fecha: dd/MM/yyyy HH:mm:ss -> Descripción: Descripción del evento.”, ...]

6.3.2. Filtrar logs

Tipos de eventos posibles:

- type = 0 : Creación de estudiantes.
- type = 1 : Creación de docentes.
- type = 2 : Creación de cursos.
- type = 3 : Eliminación de cursos.
- type = 4 : Inscripción de un estudiante a un curso.
- type = 5 : Corrección de una tarea.

URI	api/logs/{type}
Verbo	GET
Headers	Content-Type: application/json
Body	
Respuesta	[“Fecha: dd/MM/yyyy HH:mm:ss -> Descripción: Descripción del evento.”, ...]

6.4. Student Tasks Controller

Aquí se detalla la funcionalidades relacionadas con las tareas de los estudiantes, como obtener las pendientes de corrección y corregirlas. Para realizar las operaciones de `StudentTasksController`, tenemos como premisa, que el docente esté loggeado, esto lo validamos solicitando a quien desée realizar las operaciones, que nos envíe el token que recibió al iniciar sesión como docente.

6.4.1. Obtener tareas pendientes de corrección

URI	api/tasks/{token}
Verbo	GET
Headers	Content-Type: application/json
Body	
Respuesta	[“Curso: Redes, Tarea: Obg2 [Calificación máxima: 30], Estudiante: 203357 (mail@gmail.com)”, ...]

6.4.2. Corregir tareas:

URI	api/tasks/{token}/{courseName}/{taskName}/{studentNumber}/{score}
Verbo	GET
Headers	Content-Type: application/json
Body	
Respuesta	“Estudiante corregido correctamente.”

7. Supuestos

Los supuestos que utilizamos en el desarrollo de la aplicación están detallados a continuación:

- No pueden existir 2 cursos con el mismo nombre.
- El puntaje máximo de un curso es de 100.
- La suma de todas las tareas de un curso no pueden superar 100.
- Para que un estudiante suba una tarea a un curso, el administrador debió haberla creado en el mismo.
- Las notificaciones se verifican cada 30 segundos.
- Existe un perfil "administrador" que es el encargado de realizar todas las tareas del lado del servidor.
  - nombre : admin
  - contraseña : admin
- Los números de estudiante empiezan a contar en 200000, por lo que el primer estudiante registrado tendrá ese número, y luego irá incrementando automáticamente de 1 en 1.
- Al subir una tarea, el archivo queda en la carpeta `‘‘Release’’` del servidor con el siguiente nombre:
  nombre_del_curso‘-’numero_de_estudiante‘-’nombre_de_la_tarea‘.’extensión