

Servidor Intermedio

Por: Joseph Valverde,
Esteban Porras y Johana Wu

INDICE



01

Propósito



02

Clases en común



03

Descubrimiento con UDP

INDICE



04

Servidor Intermediario



05

Servidor de Piezas



06

Cliente

01

PROPÓSITO

LA IDEA DETRÁS



La Idea detrás



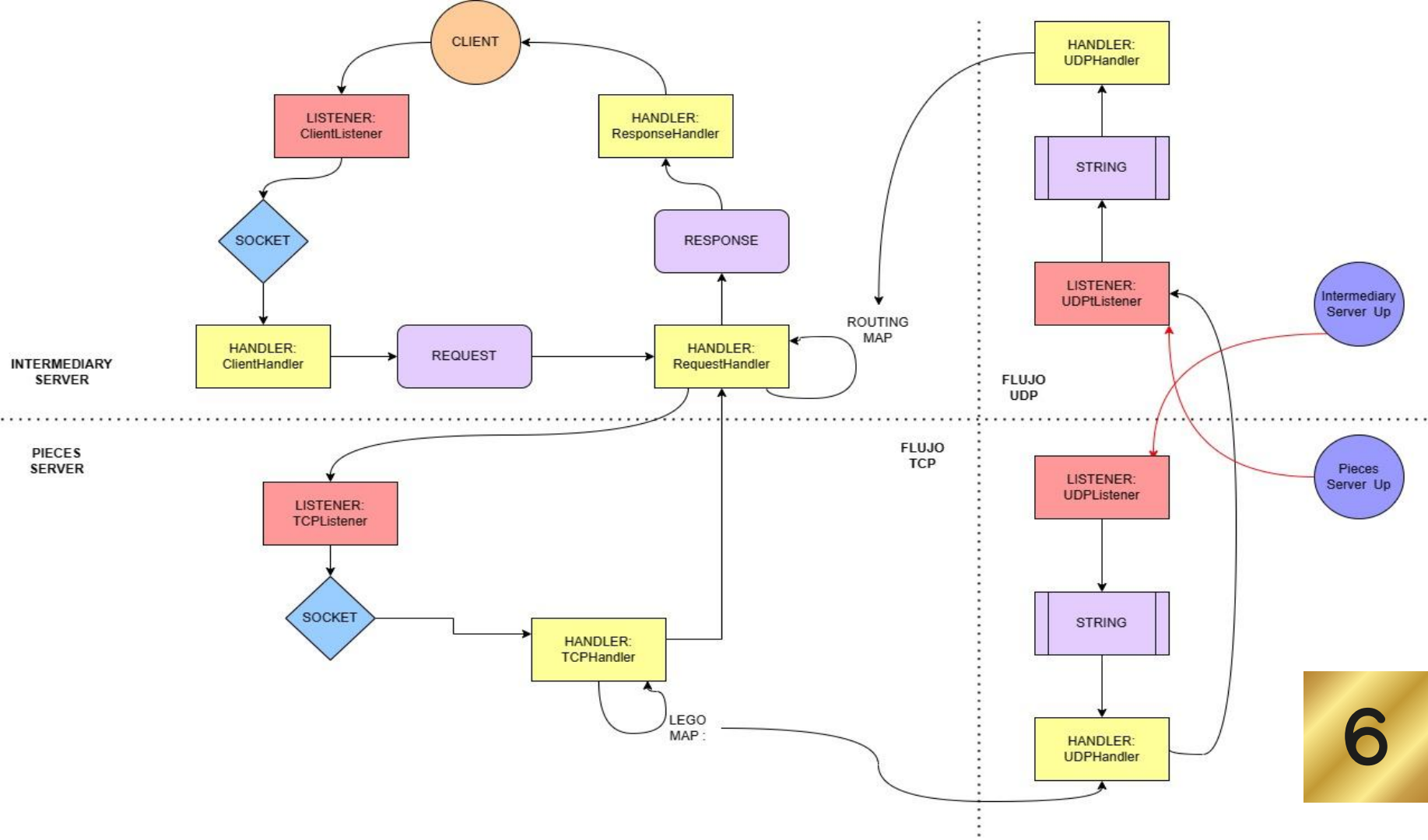
**Propósito
del servidor**



**Interconexión
con otros
servidores**



Islas



02

Clases en Común

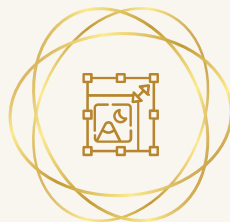
Socket, Hilos, Handlers y Listeners

7

Clases en Común

SOCKET

Interacción con
sockets



THREAD

Clase para manejar
hilos.

HANDLER

Maneja y procesa
elementos de una cola.



LISTENER

Escucha y encola
elementos de un socket.

03

Descubrimiento con UDP

9

```
void handleSingle(handlingData) {  
    // Extracción del código y la dirección IP/puerto  
    // Realización de acciones basadas en el código  
    recibido  
}
```

```
void insertFigures(buffer, ip, port) {  
    // Procesamiento de los datos de las figuras  
    recibidas  
    // Inserción de las figuras en el RoutingMap  
    junto con la IP y el puerto  
}
```

```
switch(code) {  
  case LEGO_PRESENT:  
    // Acciones cuando el código es LEGO_PRESENT  
    break;  
  case LEGO_RELEASE:  
    // Acciones cuando el código es LEGO_RELEASE  
    break;  
}
```



04

Servidor Intermediario

12

Handlers & Colas

```
std::vector <ClientHandler*> handleClientConnections;  
std::vector <UDPHandler*> handleUDP;  
std::vector <RequestHandler*> handleRequest;  
std::vector <ResponseHandler*> handleResponses;
```

```
Queue <std::shared_ptr<Socket>> ClientRequests;  
Queue <std::shared_ptr<std::vector<char>>> UDPRequests;  
Queue <std::shared_ptr<Request>> RequestQueue;  
Queue <std::shared_ptr<Response>> responseQueue;
```

Mapa de Enrutamiento

```
std::pair<std::string, int>& operator[](figure) {
```

```
    // Acceso concurrente al mapa
```

```
    this->access.lock(); // Bloquea el acceso al mapa
```

```
    std::pair<std::string, int>& pair = myRouteMap[figure]; // Accede  
    a la figura y su información asociada
```

```
    this->access.unlock(); // Desbloquea el acceso al mapa  
    return pair; // Devuelve una referencia a la figura y su  
    información asociada
```

```
void insert( figureStructure ) {
```

```
    // Acceso concurrente al mapa
```

```
    this->access.lock(); // Bloquea el acceso al mapa
```

```
    myRouteMap.insert(figureStructure); // Inserta una nueva  
    figura y su información asociada en el mapa
```

```
    this->access.unlock(); // Desbloquea el acceso al mapa
```

```
}
```

```
void showMap() {
```

```
    // Acceso concurrente al mapa
```

```
    this->access.lock(); // Bloquea el acceso al mapa
```

```
    for (auto& pair : myRouteMap) {
```

```
        std::cout << pair.first << " -> " << pair.second.first << ", " <<  
pair.second.second << std::endl;
```

```
    } // Recorre el mapa y muestra cada figura y su información  
    asociada en la consola
```

```
    this->access.unlock(); // Desbloquea el acceso al mapa
```

```
}
```


HTTP

- Manejo básico del protocolo HTTP
- El servidor HTTP lee las solicitudes enviadas por los clientes y genera la respuesta.
- Permite la comunicación entre clientes y servidores.
- `http://localhost:2020/`

ESJOJO

¡Bienvenido al servidor intermedio!

Elegir ▾

17.1

HTTP

- Se buscan identificadores en los requests para diferenciar el manejo dentro del servidor

```
lego/index.php
```

```
(.jpg)|(.png)|(.jpeg)
```

```
assemble
```

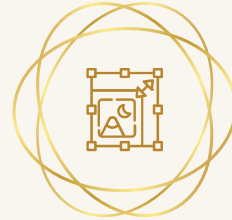
```
favicon.ico
```

Handlers .hpp

Clases - Handlers

ClientHandler

Recibe las solicitudes de los clientes y las envía a `RequestHandler`



RequestHandler

Procesa la solicitud y genera una respuesta adecuada

ResponseHandler

Se encarga de enviar la respuesta al cliente a través del socket correspondiente





Intermediary Server .hpp

Clases – IntermediaryServer.hpp

start

Configura y pone en marcha los componentes necesarios para la recepción y manejo de solicitudes de los clientes.



stopServer

Detiene el servidor intermediario, deteniendo los oyentes.

broadcastPresence

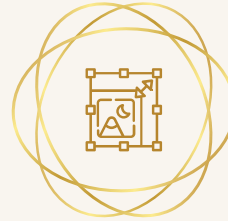
Envía un mensaje de difusión para anunciar la presencia del servidor intermediario en la red.



Clases - IntermediaryServer.hpp

broadcast

Mensaje de difusión a través de sockets UDP para alcanzar diferentes redes.



broadcastIsland

Configura la dirección de difusión específica de una red y envía el mensaje de difusión a través del socket UDP correspondiente.

broadCastOnSamePC

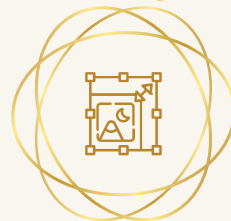
Envía el mensaje de difusión a través del socket UDP en la misma computadora.



Listeners.hpp



Clases - Listeners.hpp



ClientListener

Escucha y acepta conexiones de clientes en un socket, realiza configuraciones de seguridad y agrega las conexiones a una cola para su procesamiento posterior.

UDPListener

Escucha y recibe mensajes UDP en un socket, guarda los datos recibidos en un búfer y los devuelve como un objeto `std::vector<char>` para su posterior procesamiento.

05

Servidor de Piezas

25

PiecesServe r.hpp

ATRIBUTOS

UDPListener* listenUDP;

TCPListener* listenTCP;

std::vector<UDPHandler*> handleUDP;

std::vector<TCPHandler*> handleTCP;

Queue<std::shared_ptr<std::vector<char>>> UDPSockets;

Queue<std::shared_ptr<Socket>> TCPSockets;

Socket* connectionSocket;

Socket* communicationsSocket;

std::string legoSourceFileName;

LegoMap myFigures;

LegoMap myFigures

```
struct Lego {  
    std::string imageFigure;  
    std::string description;  
    size_t amount;
```

```
    Lego(std::string imageFigure = "", std::string description = "", size_t amount = 0) :  
        imageFigure(imageFigure),  
        description(description), amount(amount) {}  
};
```

```
typedef std::map<std::string, std::pair<std::string,  
    std::vector<Lego>>> LegoMap;
```

Formato de legoSourceFile

Lego source File :: group ESJOJO

Figura1

Figura1.jpg

Pieza1_Figura1

Pieza1_Figura1.jpg

Cantidad_Pieza1_Figura1

Pieza2_Figura1

Pieza2_Figura1.jpg

Cantidad_Pieza2_Figura1

*

Figura2

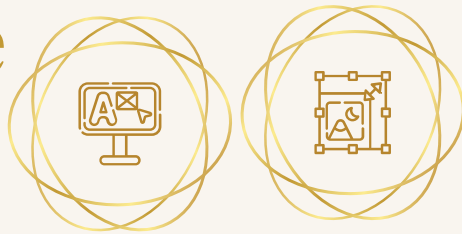
...

26.3

Funciones del Servidor de Piezas

readLegoSourceFile

Lee un archivo de origen de Lego y carga los datos en la estructura myFigures.



stopServer

Detiene el servidor de piezas y envía un mensaje UDP de liberación.

broadcastPresence


Envía información sobre las figuras de Lego disponibles mediante un mensaje UDP.



06


Cliente

27



El cliente se integra fácilmente con nuestro servidor, el cual es compatible con la página de Chiki.

Solo se requiere realizar ajustes en la configuración de IP y puerto para asegurar su correcto funcionamiento. Utiliza solicitudes HTTP GET para obtener información sobre figuras y piezas de Lego.



En lugar de conectarse al servidor de Chiki, establece conexión con nuestro servidor, permitiendo una comunicación efectiva y la recepción de los datos necesarios para su operación.

07



BONUS: Envío
de imágenes ✨







Requerimientos:

1. Responder a los request de SRC con un stream de bytes de imagen.
2. No cambiar comunicación entre servidores para evitar problemas con el protocolo.
3. Mandar grandes cantidades de datos.

Soluciones

1.

- Identificar un nuevo request.
 - Reconocer path de SRC como pedido de imagen.

8	2x2 piel	
4	2x1 blanco	
2	1x1 piel	
2	1x4 blanco	
10	1x4 piel	
		
Total de piezas para armar esta figura		128

31

Soluciones

2.

- Utilizar mismo LEGO_REQUEST para pedir imagenes.
- Utilizar mismo LEGO_RESPONSE para enviar.
- Servidor de piezas sabe diferenciar entre ambos.
- Otros servidores de piezas no saben diferenciar, por lo que retornarán error
- Servidor intermedio sabe manejar ese error.

Soluciones

3.

- Buffers pueden adaptar su tamaño dinámicamente.
- SSL tiene tamaño máximo de envío de 16 kilobytes. Identificar un nuevo request.
 - Reconocer path de SRC como pedido de imagen.
- Extra: para mostrar icono en el tab se responde al request de navegador: “favicon.ico”, pero el servidor intermedio posee la imagen, no el servidor de piezas.





08

Demostración

Ejecución del código

¡GRACIAS!

