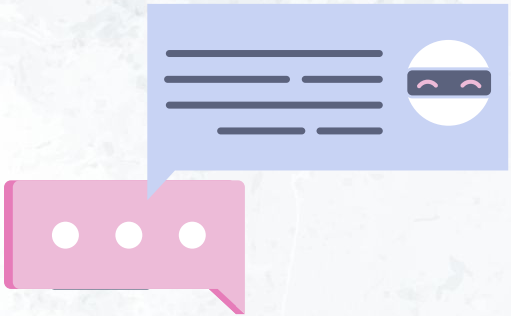
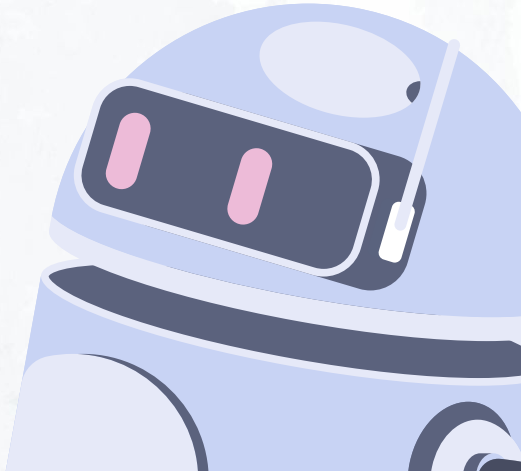
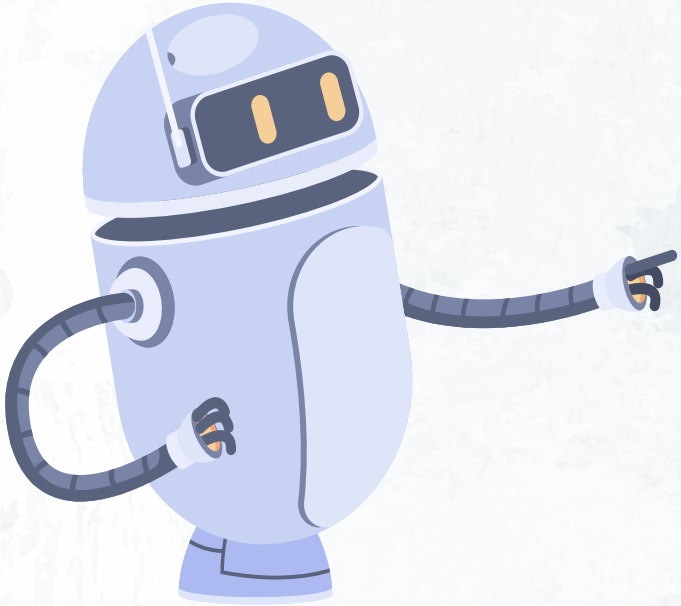
Servidor de Piezas



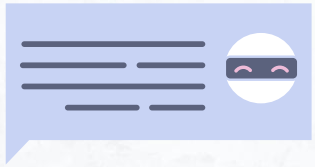
Por
-Joseph Valverde
-Johana Wu
-Esteban Porras



Índice



01 →



FUNCIONAMIENTO

Propósito del servidor

¿Cómo realiza la conexión?

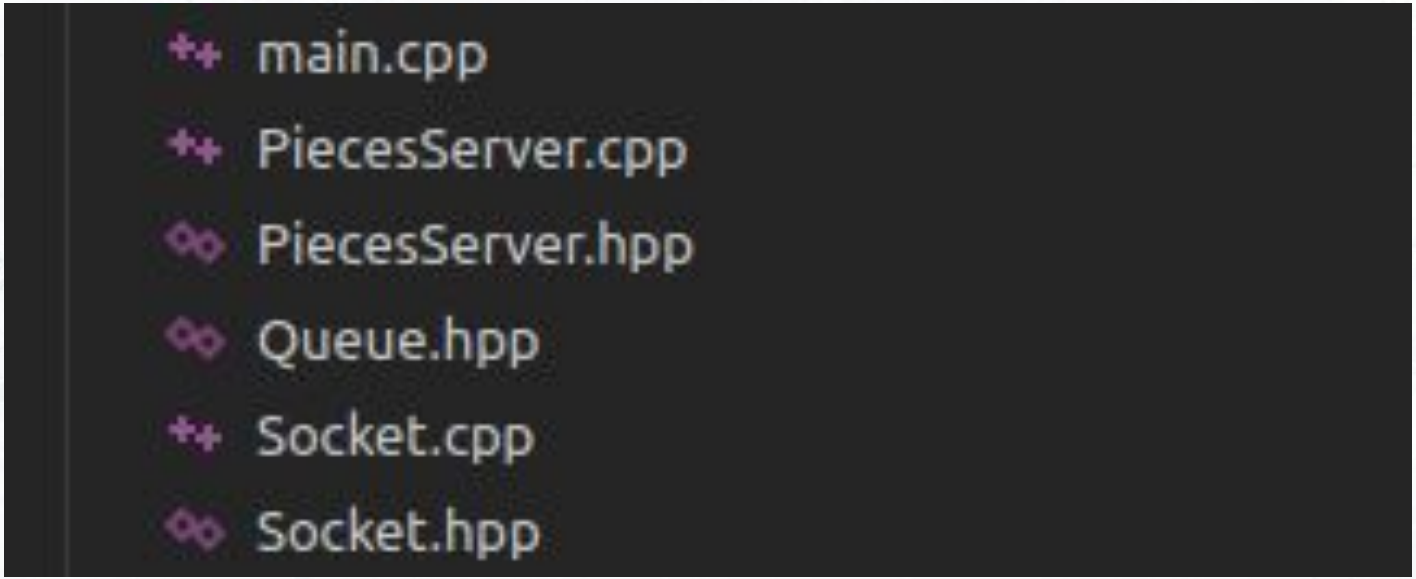
Puertos utilizados

Futuras implementaciones

02 →

METODOLOGÍA

Clases implementadas



Main.cpp

```cpp
int main (int argc, char** argv) {
  (void) argc;
  (void) argv;

  // Set signal handlers for SIGINT, SIGTERM and SIGSTOP signals
  signal(SIGINT, signal_handler);
  signal(SIGTERM, signal_handler);
  signal(SIGSTOP, signal_handler);

  // Read Lego source file
  PiecesServer::getInstance().readLegoSourceFile();

  // Read Lego source file named "legoDragonTest.txt"
  PiecesServer::getInstance().readLegoSourceFile("legoDragonTest.txt");

  // Start the server
  PiecesServer::getInstance().startServer();

  return EXIT_SUCCESS; /** Return exit status indicating successful program completion */
}
```

PiecesServer.cpp


```cpp
void PiecesServer::startServer() {
  Socket* client;

  // Bind the client socket to the CLIENT_PORT and initialize
  this->clientSocket->Bind(CLIENT_PORT);
  this->clientSocket->Listen(5);
  this->clientSocket->SSLInitServer("esjojo.pem", "key.pem");

  // Bind the browser socket to the BROWSER_PORT
  this->browserSocket->Bind(BROWSER_PORT);
  this->browserSocket->Listen(5);

  std::cout << "listening" << std::endl;

  // Create a thread for listening to browser connections
  this->browserListenThread =
      std::shared_ptr<std::thread>
      (new std::thread(listenBrowserConnections, this));

  // make thread that processes browser requests
  this->browserRequestHandler =
      std::shared_ptr<std::thread>
      (new std::thread(processBrowserRequests, this));

  // make thread that processes client requests
  this->clientRequestHandler =
      std::shared_ptr<std::thread>
      (new std::thread(processClientRequests, this));
```

```
  // make thread that listens to cin

  // look for connectionimage.png requests on socket
  while (true) {
    std::cout << "Listening to client connections" << std::endl;

    client = this->clientSocket->Accept();

    if ((int)(size_t)client == -1 || client == nullptr || this->closing) {
      this->clientQueue.push(nullptr);
      break;
    }
    // queue the requests
    this->clientQueue.push(client);
  }

  if (!this->closing) {

  }

  // Join the client request handler thread
  this->clientRequestHandler->join();
  std::cout << "Client threads joined" << std::endl;

  // Close the client socket and deallocate the memory
  this->clientSocket->Close();
  delete this->clientSocket;
}
```

PiecesServer.hpp

```
static void processBrowserRequests(PiecesServer* server) {
  Socket* client = nullptr;

  // Continuously process browser requests until signaled to stop
  while (true) {
    client = server->browserQueue.pop(); // Pop a client from the browser queue

    // Check if there is an error in popping the client or if the server is closing
    if ((int)(size_t)client == -1 || client == nullptr) {
      break;
    }

    // Process the browser request for the client using the legos map
    processBrowserRequest(client, server->legos);

    std::cout << "Browser request served" << std::endl;
  }

  // Once browser requests are done, close client requests
  // Since the main thread may be stolen by a signal, it must first finish signal handling
  // before the main thread running on client listening can begin ending
  server->clientSocket->Close();
}
```

```
static void processBrowserRequest (Socket* client,
    const std::map<std::string, size_t>& legos) {
  std::cout << "Serving browser request" << std::endl;
  std::string response =
      // send header
      "HTTP/1.0 200\r\n"
      "Content-type: text/html; charset=UTF-8\r\n"
      "Server: AttoServer v1.1\r\n"
      "\r\n"

      // send html format and title
      "<!DOCTYPE html>\n"
      "<html>\n"
      "<html lang=\"en\">\n"
      "   <meta charset=\"UTF-8\"/>\n"
      "   <title>Figures Server Pieces List </title>\n"
      "   <style>body {font-family: monospace}</style>\n"
      "   <h1>Figures Server Pieces List</h1>\n"
      "<TABLE BORDER=1 BGCOLOR=\"pink\" CELLPADDING=5 ALIGN=LEFT>\n"
      "<TR> <TH> Cantidad </TH> <TH> Descripción </TH> </TR>\n";
```

```
// add figures and their amounts to the table
for (auto it = legos.begin();
    it != legos.end();
    it++) {
  if (it->second != 0) {
    response.append("<TR><TD ALIGN=center> ");
    response.append(std::to_string(it->second));
    response.append(
        "</TD>\n"
        "<TD ALIGN=center> ");
    response.append(it->first);
    response.append(
      "</TD>\n"
      "</TR>\n");
  }
}
```

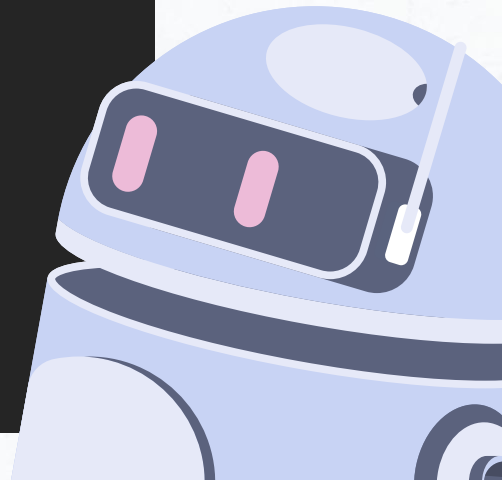
```
  // close table and html doc
  response.append(
      "</TR></TABLE>\n"
      "</html>");

  // send all bytes
  client->Write(
      response.c_str(),
      response.size()
      );
}
```

```
static void listenBrowserConnections(PiecesServer* piecesServer) {
  Socket* client;

  // look for connection requests on socket
  while (true) {
    std::cout << "Listening to browser connections" << std::endl;

    // Accept a new browser connection
    client = piecesServer->browserSocket->Accept();

    // Check if the client connection is valid or if the server is closing
    if ((int)(size_t)client == -1 || client == nullptr || piecesServer->closing) {
      std::cout << "Ending browser connections thread" << std::endl;
      piecesServer->browserQueue.push(nullptr);
      break;
    }

    std::cout << "Browser connection accepted\n" << std::endl;

    // Queue the client connection for further processing
    piecesServer->browserQueue.push(client);
  }
}
```

```
static void processClientRequests(PiecesServer* server) {
  Socket* client = nullptr;

  // Continuously process client requests until signaled to stop
  while (true) {
    client = server->clientQueue.pop(); // Pop a client from the client queue

    // Check if there is an error in popping the client or if the server is closing
    if ((int)(size_t)client == -1 || client == nullptr) {
      break;
    }

    client->SSLCreate(server->clientSocket); // Create an SSL context for the client
    client->SSLAccept(); // Perform SSL handshake with the client
    processClientRequest(client, server->legos); // Process the client request
  }
}
```

```
static void processClientRequest (Socket* client,
    std::map<std::string, size_t>& legos) {

  char response[2];
  memset(response, 0, 2);
  response[0] = '0';

  std::vector<std::pair<std::string, size_t>> requestedPieces;

  // recibir piezas
  processRequest(client, requestedPieces);

  // amount of pieces found
  size_t piecesFountAmount = 0;

  // check if all pieces are available
  for (size_t piece = 0; piece < requestedPieces.size(); piece++) {
    if (legos[requestedPieces[piece].first] >=
        requestedPieces[piece].second) {
      piecesFountAmount++;
    }
  }
```

```
  // if available take out the pieces
  if (piecesFountAmount == requestedPieces.size()) {
    // check if all pieces are available
    for (size_t piece = 0; piece < requestedPieces.size(); piece++) {
      if (legos[requestedPieces[piece].first] != 0) {
        legos[requestedPieces[piece].first] -=
          requestedPieces[piece].second;
      }
    }

    // set response to positive
    response[0] = '1';
  }

  // send all bytes
  client->SSLWrite(
      response,
      2
      );

}
```



Queue.hpp


```
void push(const dataType& data) {
  this->canAccess.lock(); // Acquire the lock to access the queue
  this->queue.push(data); // Push the data to the queue
  this->canAccess.unlock(); // Release the lock
  sem_post(&this->canConsume); // Signal the semaphore to unblock consumers
}

dataType pop() {
  sem_wait(&this->canConsume); // Wait until there is an element available in the queue
  this->canAccess.lock(); // Acquire the lock to access the queue
  dataType& resultData = this->queue.front(); // Get the front element of the queue
  this->queue.pop(); // Remove the front element from the queue
  this->canAccess.unlock(); // Release the lock
  return resultData; // Return the popped element
}
```

03 →

DEMOSTRACIÓN

¡GRACIAS POR SU ATENCIÓN!