```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <ctime>
#include <iomanip>
#include <algorithm>
#include <winsock2.h>

bool recibirDatos(SOCKET clientSocket, char* buffer, int bufferSize);

std::string leerArchivo(const std::string& nombreArchivo);

std::pair<bool, std::string> verificarCredenciales(const std::string& usuario, const std::string& contrasena);

void registrarLog(const std::string& mensaje);

std::string fechaYHora();

std::string traduccion(SOCKET clientSocket);

std::string buscarTraduccion(const std::string& palabra);

std::string nuevaTraduccion(SOCKET clientSocket);  // Esta es para PEDIR al cliente una nueva traducción

bool agregarNuevaTraduccion(const std::string& nuevaTraduccion); // Esta es para METER EN EL ARCHIVO la nueva traducción

std::string verRegistroActividades(SOCKET clientSocket); // Manda por varios buffer el registro completo

std::string administrarUsuarios(SOCKET clientSocket);
// Agrega un usuario al archivo credenciales.txt
bool registrarNuevoUsuario(const std::string& usuario, const std::string& contrasena, const std::string& rol, int intentos);

bool usuarioExiste(const std::string& usuario); // Función para verificar si un usuario ya está registrado

std::string obtenerUsuariosBloqueados(); // Devuelve un string con los usuarios que estan bloqueados

std::string desbloquearUsuario(const std::string& usuario); // le asigna 0 intentos a un usuario con mas de 3

std::string menuConsulta();

std::string menuAdmin();

std::string mostrarMenu(std::string rol);

int main() {
    std::string ip;
    int puerto;
    std::cout << "Ingrese la dirección IP del servidor: ";
    std::cin >> ip;
    std::cout << "Ingrese el puerto del servidor: ";
    std::cin >> puerto;
    std::cin.ignore();

    std::string mensajeLog="";
    std::string response;
    WSADATA wsData;
```

```
59      if (WSAStartup(MAKEWORD(2, 2), &wsData) != 0) {
60          std::cerr << "Error al inicializar Winsock" << std::endl;
61          return -1;
62      }
63
64      SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
65      if (serverSocket == INVALID_SOCKET) {
66          std::cerr << "Error al crear el socket del servidor" << std::endl;
67          WSACleanup();
68          return -1;
69      }
70
71      sockaddr_in serverAddr;
72      serverAddr.sin_family = AF_INET;
73      serverAddr.sin_port = htons(puerto);
74      serverAddr.sin_addr.s_addr = inet_addr(ip.c_str());
75
76
77      int reuseAddr = 1;
78      if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, reinterpret_cast<const
char*>(&reuseAddr), sizeof(reuseAddr)) == SOCKET_ERROR) {
79          std::cerr << "Error al establecer SO_REUSEADDR" << std::endl;
80          closesocket(serverSocket);
81          WSACleanup();
82          return -1;
83      }
84
85      if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
86          std::cerr << "Error al vincular el socket a la direccion" << WSAGetLastError
() << std::endl;
87          closesocket(serverSocket);
88          WSACleanup();
89          return -1;
90      }
91
92      // Obtener el puerto de escucha
93      sockaddr_in boundAddr;
94      int boundAddrLen = sizeof(boundAddr);
95      if (getsockname(serverSocket, (sockaddr*)&boundAddr, &boundAddrLen) ==
SOCKET_ERROR) {
96          std::cerr << "Error al obtener el puerto de escucha" << WSAGetLastError() <<
std::endl;
97          closesocket(serverSocket);
98          WSACleanup();
99          return -1;
100     }
101
102     unsigned short puertoDeEscucha = ntohs(boundAddr.sin_port);
103
104     if (listen(serverSocket, 5) == SOCKET_ERROR) {
105         std::cerr << "Error al escuchar por conexiones entrantes" << std::endl;
106         closesocket(serverSocket);
107         WSACleanup();
108         return -1;
109     }
110     mensajeLog = "============================\n =======Inicia
Servidor=======\n============================";
111     registrarLog(mensajeLog);
112
113     std::cout << "Esperando conexiones entrantes..." << std::endl;
114     // Acá guardo el puerto de escucha
115     mensajeLog = fechaYHora() + ": Socket creado. Puerto de escucha: " + std::
to_string(puertoDeEscucha);
116     registrarLog(mensajeLog);
117
```

```
118      while (true) { // Bucle externo para esperar nuevas conexiones
119          sockaddr_in clientAddr;
120          int clientAddrSize = sizeof(clientAddr);
121          SOCKET clientSocket = accept(serverSocket, (sockaddr*)&clientAddr, &
clientAddrSize);
122          if (clientSocket == INVALID_SOCKET) {
123              std::cerr << "Error al aceptar la conexion entrante" << std::endl;
124              closesocket(serverSocket);
125              WSACleanup();
126              return -1;
127          }
128
129          std::cout << "Cliente conectado" << std::endl;
130
131          bool credencialesPedidas = false; // Creo el boolean para que las
credenciales no esten en el while
132          std::string usuario;
133          std::string rolUsuario;
134          char buffer[1024];
135          while (true) {
136
137              if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
138                  break; // Manejar error o desconexión del cliente
139              }
140
141              std::cout << "Cliente: " << buffer << std::endl;
142              std::string mensajeCliente(buffer);
143
144              if (!credencialesPedidas) { ///ESTE IF ES PARA LA PRIMERA VEZ, O SEA
PARA INICIAR SESION, SU ELSE ES PARA ELEGIR OPCIONES
145              // Parsear el mensaje para obtener usuario y contraseña
146                  size_t pos = mensajeCliente.find('|');
147                  if (pos == std::string::npos) {
148                      std::cerr << "Mensaje invalido del cliente" << std::endl;
149                      continue;
150                  }
151                  usuario = mensajeCliente.substr(0, pos);
152                  std::string contrasena = mensajeCliente.substr(pos + 1);
153
154                  // Verificar las credenciales
155
156                  std::pair<bool, std::string> credenciales = verificarCredenciales(
usuario, contrasena);
157                  rolUsuario = credenciales.second; // Esto es para filtrar las
opciones según el rol
158
159                  if (credenciales.first) {
160                      response = "Acceso concedido como " + credenciales.second +
mostrarMenu(rolUsuario);
161                      credencialesPedidas = true;
162                      mensajeLog = fechaYHora() + ": Inicio de sesion - usuario: " +
usuario;  // Esto va a registro.log
163                      registrarLog(mensajeLog);
164                  } else {
165                      response = credenciales.second;
166                      send(clientSocket, response.c_str(), response.size(), 0);
167                      break;
168                  }
169                  send(clientSocket, response.c_str(), response.size(), 0);
170              } else { ///ACA SE ELIGEN OPCIONES Y SE HACEN LAS FUNCIONES NECESARIAS
171                  if (mensajeCliente == "1") { // Traducir palabra que recibe el
servidor
172                      response = traduccion(clientSocket) + mostrarMenu(rolUsuario);
173                  } else if (mensajeCliente == "2" && rolUsuario == "ADMIN") { //
Agregar una traducción
174                      response = nuevaTraduccion(clientSocket) + mostrarMenu(
```

```
rolUsuario);
                } else if (mensajeCliente == "3" && rolUsuario == "ADMIN") { //
Usuarios
                    response = administrarUsuarios(clientSocket) + mostrarMenu(
rolUsuario);
                } else if (mensajeCliente == "4" && rolUsuario == "ADMIN") { // Ver
registro de actividades
                    response = verRegistroActividades(clientSocket) + mostrarMenu(
rolUsuario);
                } else if (mensajeCliente == "5") { // Cerrar Sesión
                    break; // Aqui cierra sesión
                } else { // Opción no válida
                    response = "Opcion no valida" + mostrarMenu(rolUsuario);
                }
                //response = "Mensaje recibido por el servidor";
                send(clientSocket, response.c_str(), response.size(), 0);
            }
            //send(clientSocket, response.c_str(), response.size(), 0);
        }
        mensajeLog = fechaYHora() + ": Cierre de sesion - usuario: " + usuario;
        registrarLog(mensajeLog);
        closesocket(clientSocket); // Acá cierro el socket dentro del while, asi
puedo...

        std::cout << "Esperando nuevas conexiones..." << std::endl; // Volver a
esperar nuevas conexiones
    }

    closesocket(serverSocket);
    WSACleanup();

    return 0;
}

bool recibirDatos(SOCKET clientSocket, char* buffer, int bufferSize) {
    memset(buffer, 0, bufferSize);
    int bytesReceived = recv(clientSocket, buffer, bufferSize - 1, 0);
    if (bytesReceived == SOCKET_ERROR) {
        std::cerr << "Error al recibir datos del cliente" << std::endl;
        return false;
    } else if (bytesReceived == 0) {
        std::cout << "Cliente desconectado" << std::endl;
        return false;
    }
    return true;
}

std::string leerArchivo(const std::string& nombreArchivo) { // Le mando por
parámetro el nombre del archivo
    std::ifstream archivo(nombreArchivo); // Leer archivo
    std::string contenido; // String para cargar los datos (ACA VAN A ESTAR LOS
STRINGS PARA COMPARAR)

    if (archivo.is_open()) { // Si el archivo está abierto
        std::string linea; // Línea temporal
        while (std::getline(archivo, linea)) { // Acá igualo la línea temporal a la
siguiente línea del archivo (SE REEMPLAZA)
            contenido += linea + "\n"; // Le meto al string para cargar datos la
línea temporal
        }
        archivo.close();
    } else {
        std::cerr << "Error al abrir el archivo" << std::endl;
    }

    return contenido;
```

```cpp
 }

 std::pair<bool, std::string> verificarCredenciales(const std::string& usuario, const std::string& contrasena) {
      std::string contenidoArchivo = leerArchivo("credenciales.txt");

      std::istringstream archivoStream(contenidoArchivo); // Acá guardo las credenciales para estructurar funciones
      std::string linea;
      std::vector<std::string> lineas;

      bool usuarioBloqueado = false;
      bool usuarioEncontrado = false;
      bool accesoConcedido = false;

      std::string rolArchivoReturn = "";// Declaro rol para que me lo tome el return

      while (std::getline(archivoStream, linea)) { // Llamo una línea de las credenciales
          std::istringstream lineaStream(linea); // Lunciones para la línea
          std::string usuarioArchivo, contrasenaArchivo, rolArchivo, intentosArchivo; // Strings usuario|contraseña|rol|intentos
          std::getline(lineaStream, usuarioArchivo, '|');
          std::getline(lineaStream, contrasenaArchivo, '|');
          std::getline(lineaStream, rolArchivo, '|');
          std::getline(lineaStream, intentosArchivo);
          if (usuarioArchivo == usuario) {
              usuarioEncontrado = true;
              int intentos = std::stoi(intentosArchivo); // Convertir el valor a entero

              if (intentos >= 3) {
                  usuarioBloqueado = true;
                  rolArchivoReturn = "usuario bloqueado";
              } else if (contrasenaArchivo == contrasena) {
                  intentos = 0;
                  accesoConcedido = true;
                  rolArchivoReturn = rolArchivo;
              } else if(intentos == 2){
                  rolArchivoReturn = "Datos de usuario incorrectos\nUSUARIO BLOQUEADO";
                  intentos++;
              } else {
                  rolArchivoReturn = "Datos de usuario incorrectos";
                  intentos++;
              }

              linea = usuarioArchivo + "|" + contrasenaArchivo + "|" + rolArchivo + "|" + std::to_string(intentos); // Paso a int intentos
          }

          lineas.push_back(linea); // Agregar la línea a la lista
      }

      if (!usuarioEncontrado) {
          std::cout << "Usuario no encontrado" << std::endl;
      }

      if (!usuarioBloqueado && !accesoConcedido) {
          std::ofstream archivo("credenciales.txt"); // Escribimos en el archivo
          for (const std::string& lineaModificada : lineas) { // Recorro cada línea modificada
              archivo << lineaModificada << "\n"; // La imprimo en el archivo
          }
      }
      // Devolver un par (bool, string) donde el bool indica si las credenciales son
```

```
válidas
288         // y el string contiene el rol del usuario
289         return std::make_pair(usuarioEncontrado && accesoConcedido, rolArchivoReturn);
290     }
291
292     void registrarLog(const std::string& mensaje) {
293         std::ofstream archivoLog("registro.log", std::ios::app); // Abre el archivo en
modo de adjuntar
294         if (archivoLog.is_open()) {
295             archivoLog << mensaje << std::endl; // Escribe el mensaje en el archivo log
296             archivoLog.close(); // Cierra el archivo
297         } else {
298             std::cerr << "Error al abrir el archivo de registro.log" << std::endl;
299         }
300     }
301
302     std::string fechaYHora(){
303         // Obtener la fecha y la hora actual del sistema
304         std::time_t tiempoActual = std::time(nullptr);
305         std::tm* tiempoInfo = std::localtime(&tiempoActual);
306
307         // Formatear la fecha y la hora como una cadena
308         std::stringstream ss;
309         ss << std::put_time(tiempoInfo, "%Y-%m-%d_%H:%M");
310         std::string fechaYHora = ss.str();
311
312         return fechaYHora;
313     }
314
315     std::string traduccion(SOCKET clientSocket) {
316         // Puedes recibir datos adicionales del cliente y enviar respuestas aquí
317         char buffer[1024];
318         // Pedir la palabra del cliente
319         send(clientSocket, "Ingrese una palabra en ingles ", strlen("Ingrese una palabra
en ingles "), 0);
320
321         // Recibe la palabra en inglés del cliente
322         if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
323             return "ERROR al recibir datos";
324         }
325         // Ahora con la palabra en mano, la mando en la funcion de buscar traduccion
326         std::string palabra=buffer;
327         if (palabra == "/salir") return "";
328
329         // Convertir la palabra a minúsculas
330         std::transform(palabra.begin(), palabra.end(), palabra.begin(), ::tolower);
331
332
333         // La traducción no existe en el archivo
334         std::string resultado = buscarTraduccion(palabra);
335         if (resultado.empty()) {
336             return "No fue posible encontrar la traducción para: " + palabra;
337         }
338
339         // Realiza la traducción y envía la respuesta al cliente
340         return resultado;
341     }
342
343     std::string buscarTraduccion(const std::string& palabra) {
344         const std::string archivoTraducciones = "traducciones.txt";
345
346         std::ifstream archivo(archivoTraducciones); // Abro el archivo en modo lectura
347
348         if (!archivo.is_open()) {
349             return "Error: No se pudo abrir el archivo";
350         }
```

```
351
352     std::string linea;
353     while (std::getline(archivo, linea)) {
354         std::istringstream ss(linea);
355         std::string palabraEnIngles, traduccion;
356
357         if (std::getline(ss, palabraEnIngles, ':') && std::getline(ss, traduccion))
358 {
359             // Eliminar espacios en blanco al principio y al final de las cadenas
360             palabraEnIngles.erase(0, palabraEnIngles.find_first_not_of(" \t\r\n"));
361             palabraEnIngles.erase(palabraEnIngles.find_last_not_of(" \t\r\n") + 1);
362
363             traduccion.erase(0, traduccion.find_first_not_of(" \t\r\n"));
364             traduccion.erase(traduccion.find_last_not_of(" \t\r\n") + 1);
365             std::cout << palabra << std::endl;
366             // Comparo con la palabra qwue mandé por parametro
367             if (palabraEnIngles == palabra) {
368                 return palabra + " en ingles es " + traduccion + " en español.";
369             }
370         }
371     }
372
373     return "No se encontro traduccion para la palabra " + palabra;
374 }
375
376 std::string nuevaTraduccion(SOCKET clientSocket) {
377     char buffer[1024];
378     // Pedir la nueva traducción al cliente
379     send(clientSocket, "Nueva traduccion palabraEnIngles:traduccionEnEspanol: ",
strlen("Nueva traduccion palabraEnIngles:traduccionEnEspanol: "), 0);
380
381     if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
382         return "ERROR al recibir datos";
383     }
384
385     std::string nuevaTraduccion=buffer; // Ejemplo: dog:perro
386     if (nuevaTraduccion == "/salir") return "";
387
388     // Verificar el formato de inserción
389     size_t pos = nuevaTraduccion.find(':');
390     if (pos == std::string::npos || pos == 0 || pos == nuevaTraduccion.length() - 1)
391 {
392         return "No fue posible insertar la traduccion. El formato de insercion debe
ser palabraEnIngles:traduccionEnEspanol";
393     }
394
395     std::string palabraEnIngles = nuevaTraduccion.substr(0, pos);
396     std::string traduccionEnEspanol = nuevaTraduccion.substr(pos + 1); // Separo las
palabras para buscarlas en el archivo
397
398     // Verificar si la traducción ya existe
399     if (buscarTraduccion(palabraEnIngles) == traduccionEnEspanol) {
400         return "Ya existe una traducción para " + palabraEnIngles + ": " +
traduccionEnEspanol;
401     }
402     if (agregarNuevaTraduccion(nuevaTraduccion)) {
403         return "Nueva traduccion insertada correctamente";
404     } else {
405         return "Error al insertar la nueva traduccion";
406     }
407 }
408
409 bool agregarNuevaTraduccion(const std::string& nuevaTraduccion) {
410     std::ofstream archivoTraducciones("traducciones.txt", std::ios::app);
411
412     if (!archivoTraducciones) {
```

```
411             return false;
412         }
413         std::string aMinuscula = nuevaTraduccion;
414         std::transform(aMinuscula.begin(), aMinuscula.end(), aMinuscula.begin(), ::
tolower);
415         archivoTraducciones << aMinuscula << std::endl;
416 
417         archivoTraducciones.close();
418 
419         return true;
420     }
421 
422     std::string verRegistroActividades(SOCKET clientSocket) {
423         std::ifstream archivoRegistro("registro.log");
424         if (!archivoRegistro) {
425             return "ERROR: no hay archivo de registro";
426         }
427 
428         std::string linea;
429         std::string contenidoRegistro;
430         while (std::getline(archivoRegistro, linea)) { // Acá voy agregando al string
final cada linea que lee el getline
431             contenidoRegistro += linea + '\n';
432         }
433         archivoRegistro.close();
434 
435         std::string tempContenidoRegistro;
436         while (contenidoRegistro.size()>=1023){ // Mientras el string final sea mayor al
tamaño del buffer, le resto los primeros 1024 char
437             tempContenidoRegistro = '$' + contenidoRegistro.substr(0, 1023); // Le
agrego el $ para decirle al cliente que aún hay que mostrar más registro
438             contenidoRegistro = contenidoRegistro.substr(1023);
439             send(clientSocket, tempContenidoRegistro.c_str(), tempContenidoRegistro.size
(), 0);
440         }
441         contenidoRegistro = '%' + contenidoRegistro; // Cuando se termina de mostrar
todo el registro que hay en le buffer, le aviso al cliente con un % en el primer caracter
442         return contenidoRegistro;
443     }
444 
445     std::string administrarUsuarios(SOCKET clientSocket) {
446         char buffer[1024];
447         send(clientSocket, "\nUsuarios:\na. Alta\nb. Desbloqueo\n", strlen(
"\nUsuarios:\na. Alta\nb. Desbloqueo\n"), 0);
448 
449         if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
450             return "ERROR al recibir datos"; // Manejar error o desconexión del cliente
451         }
452         std::string opcion(buffer); // "a" o "b"
453         if (opcion == "/salir") return "";
454 
455         if (opcion == "a") { // Alta
456             send(clientSocket, "Ingrese el nombre de usuario y contrasena separados por
'|' (Ejemplo: usuario|contrasena): ", strlen("Ingrese el nombre de usuario y contraseña
separados por '|' (Ejemplo: usuario|contraseña): "), 0);
457 
458             if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
459                 return "ERROR al recibir datos"; // Manejar error o desconexión del
cliente
460             }
461 
462             std::string datosUsuario(buffer);
463             if (datosUsuario == "/salir") return "";
464             size_t pos = datosUsuario.find('|');
465             if (pos == std::string::npos) {
466                 return "Error al dar de alta el nuevo usuario: datos incompletos";
```

```
467         }
468
469         std::string nuevoUsuario = datosUsuario.substr(0, pos);
470         std::string nuevaContrasena = datosUsuario.substr(pos + 1);
471         std::string mensajeError = "";
472
473         if (usuarioExiste(nuevoUsuario)){
474             mensajeError = "Error: El usuario '" + nuevoUsuario + "' ya existe.";
475             return mensajeError;
476         } else if (nuevoUsuario == "" || nuevaContrasena == ""){
477             mensajeError = "Error: Campos ingresados incorrectamente";
478             return mensajeError;
479         } else {
480             // Registrar el nuevo usuario, establecer intentos a 0, rol a CONSULTA
481             registrarNuevoUsuario(nuevoUsuario, nuevaContrasena, "CONSULTA", 0);
482             return "Usuario agregado con exito";
483
484         }
485     } else if (opcion == "b") { // Desbloqueo de Usuario
486         // Aquí debes implementar la lógica para mostrar los usuarios bloqueados y
permitir al cliente seleccionar uno para desbloquear
487         std::string usuariosBloqueados = obtenerUsuariosBloqueados();
488         if (usuariosBloqueados.empty()) {
489             return "No se encontraron usuarios bloqueados";
490         }
491         std::string listaDeUsuarios = "\nUsuarios Bloqueados: " + usuariosBloqueados
+ "\nIngrese el nombre de usuario a desbloquear: ";
492         send(clientSocket, listaDeUsuarios.c_str(), listaDeUsuarios.size(), 0);
493
494         if (!recibirDatos(clientSocket, buffer, sizeof(buffer))) {
495             return "ERROR al recibir datos"; // Manejar error o desconexión del
cliente
496         }
497
498         std::string usuarioADesbloquear(buffer);
499         if (usuarioADesbloquear == "/salir") return "";
500
501         // Realizar la lógica de desbloqueo del usuario
502         std::string desbloqueado = desbloquearUsuario(usuarioADesbloquear);
503
504         if (desbloqueado=="Ok") {
505             return usuarioADesbloquear + " desbloqueado correctamente";
506         } else if (desbloqueado == "File"){
507             return "Error al desbloquear el usuario";
508         } else if (desbloqueado == "No"){
509             return "El usuario no se encuentra bloqueado";
510         }
511     } else {
512         return "Opción no válida";
513     }
514 }
515
516 // Función para registrar un nuevo usuario en el archivo credenciales.txt
517 bool registrarNuevoUsuario(const std::string& usuario, const std::string& contrasena
, const std::string& rol, int intentos) {
518     // Abre el archivo en modo de escritura (appended)
519     std::ofstream archivoCredenciales("credenciales.txt", std::ios::app);
520     if (!archivoCredenciales) {
521         return false; // No se pudo abrir el archivo
522     }
523
524     // Escribe el nuevo usuario en el archivo con el formato deseado
525     archivoCredenciales << usuario << "|" << contrasena << "|" << rol << "|" <<
intentos << std::endl;
526
527     // Cierra el archivo
```

```
528     archivoCredenciales.close();
529
530     return true; // El usuario se registró correctamente
531 }
532
533 // Función para verificar si un usuario ya existe en el archivo credenciales.txt
534 bool usuarioExiste(const std::string& usuario) {
535     // Abre el archivo en modo de lectura
536     std::ifstream archivoCredenciales("credenciales.txt");
537     if (!archivoCredenciales) {
538         return false; // No se pudo abrir el archivo
539     }
540
541     std::string linea;
542     while (std::getline(archivoCredenciales, linea)) {
543         size_t pos = linea.find('|');
544         if (pos != std::string::npos) {
545             std::string usuarioExistente = linea.substr(0, pos);
546             if (usuarioExistente == usuario) {
547                 archivoCredenciales.close();
548                 return true; // El usuario ya existe en el archivo
549             }
550         }
551     }
552
553     archivoCredenciales.close();
554     return false; // El usuario no fue encontrado en el archivo
555 }
556
557 std::string obtenerUsuariosBloqueados() {
558     std::ifstream archivoCredenciales("credenciales.txt");
559     if (!archivoCredenciales) {
560         return "Error: No se pudo abrir el archivo de credenciales";
561     }
562
563     std::string usuariosBloqueados;
564     std::string linea;
565     while (std::getline(archivoCredenciales, linea)) {
566         size_t pos = linea.find('|');
567         if (pos != std::string::npos) {
568             std::string usuario = linea.substr(0, pos);
569             int intentos = std::stoi(linea.substr(linea.find_last_of('|') + 1)); //
Obtener intentos como entero
570
571             // Si el usuario tiene 3 intentos o más, considerarlo bloqueado
572             if (intentos >= 3) {
573                 if (!usuariosBloqueados.empty()) {
574                     usuariosBloqueados += ", ";
575                 }
576                 usuariosBloqueados += usuario;
577             }
578         }
579     }
580
581     archivoCredenciales.close();
582
583     if (usuariosBloqueados.empty()) {
584         return "";
585     }
586
587     return usuariosBloqueados;
588 }
589
590 std::string desbloquearUsuario(const std::string& usuario) {
591     std::ifstream archivoEntrada("credenciales.txt");
592     if (!archivoEntrada) {
```

```
593             return "File"; // No se pudo abrir el archivo
594         }
595         std::string retorno = "";
596         std::vector<std::string> lineas;
597         std::string linea;
598         std::string ultimoString = "";
599         int flag = 0;
600         while (std::getline(archivoEntrada, linea)) {
601             size_t pos = linea.find('|');
602             if (pos != std::string::npos) { // Si se encontró el '|'
603                 std::string nombreUsuario = linea.substr(0, pos);
604 
605                 if (nombreUsuario == usuario) {
606                     // Encontramos el usuario, ahora lo modificamos para establecer
intentos a 0
607                     size_t intentos = linea.find_last_of('|'); // Busco la posición del
último '|', significa que lo que siga despues van a ser los intentos
608                     ultimoString = linea.back();
609                     if (intentos != std::string::npos && ultimoString == "3") { // Si se
encontró el nuevo '|' y el ultimo caracter es 3
610                         // Encuentra la posición del último '|'
611                         linea.replace(intentos + 1, std::string::npos, "0");
612                         flag++;
613                     }
614                 }
615             }
616             lineas.push_back(linea);
617         }
618 
619         archivoEntrada.close();
620         if (flag == 0){
621             return "No";
622         }
623         // Volver a escribir todas las líneas en el archivo
624         std::ofstream archivoSalida("credenciales.txt");
625         if (!archivoSalida) {
626             return "File"; // No se pudo abrir el archivo
627         }
628 
629         for (const std::string& linea : lineas) {
630             archivoSalida << linea << '\n';
631         }
632 
633         archivoSalida.close();
634 
635         return "Ok";
636     }
637 
638     std::string menuConsulta() {
639         std::string menu;
640         menu += "\n";
641         menu += "Menu de Consulta:\n";
642         menu += "1. Traducir\n";
643         menu += "5. Cerrar Sesion\n";
644         return menu;
645     }
646 
647     std::string menuAdmin() {
648         std::string menu;
649         menu += "\n";
650         menu += "Menu de Administrador:\n";
651         menu += "1. Traducir\n";
652         menu += "2. Nueva Traduccion\n";
653         menu += "3. Usuarios:\n";
654         menu += "   a. Alta\n";
655         menu += "   b. Desbloqueo\n";
```

```
656         menu += "4. Ver Registro de Actividades\n";
657         menu += "5. Cerrar Sesion\n";
658         return menu;
659     }
660 
661     std::string mostrarMenu(std::string rol){
662         std::string retorno="\n";
663         if (rol == "ADMIN"){
664             retorno += menuAdmin();
665         } else if (rol == "CONSULTA"){
666             retorno += menuConsulta();
667         }
668         return retorno;
669     }
```