```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.IO;
 4  using System.Net;
 5  using System.Net.Sockets;
 6  using System.Security.Cryptography;
 7  using System.Text.RegularExpressions;
 8  using System.Threading;
 9  using System.Windows.Controls;
10
11  using static HotSpot.Modules.Common;
12  using static HotSpot.Modules.Cryptography;
13  using static HotSpot.Modules.Sql;
14
15  namespace HotSpot
16  {
17      public class AsyncServer
18      {
19          public AsyncServer(TextBox textBox)
20          {
21              _logBox = textBox;
22          }
23
24          private readonly TextBox _logBox;
25          private readonly ManualResetEvent serverTalk = new ManualResetEvent
              (false);
26          private readonly List<ClientObject> Ring = new List<ClientObject>();
27          private readonly string TransactionEndTag = "</transaction>";
28
29          public void StartListening(string note)
30          {
31              try
32              {
33                  IPEndPoint ipEndPoint = new IPEndPoint(IpAddress, PortNumber);
34                  using (Socket server = new Socket(IpAddress.AddressFamily,
                      SocketType.Stream, ProtocolType.Tcp))
35                  {
36                      server.Bind(ipEndPoint);
37                      server.Listen(MaxPendingConnections);
38
39                      while (true)
40                      {
41                          serverTalk.Reset();
42                          Log($"{note}\nWaiting for a Connection...");
43                          server.BeginAccept(new AsyncCallback(AcceptCallback),
                           server);
44                          serverTalk.WaitOne();
45                      }
46                  }
47              }
48              catch
49              {
```

```
50                  Log($"Server is now Disconnected...");
51              }
52          }
53
54          public void AcceptCallback(IAsyncResult asyncResult)
55          {
56              try
57              {
58
59                  serverTalk.Set();
60                  Socket reciever = (Socket)asyncResult.AsyncState;
61                  Socket worker = reciever.EndAccept(asyncResult);
62                  StateObject stateObject = new StateObject { workSocket =
                      worker };
63                  worker.BeginReceive(stateObject.buffer, 0,
                      StateObject.BufferSize, 0, new AsyncCallback(ReadCallback),
                      stateObject);
64              }
65              catch
66              {
67
68              }
69          }
70
71          public void ReadCallback(IAsyncResult asyncResult)
72          {
73              string transaction = string.Empty;
74
75              try
76              {
77                  StateObject stateObject = (StateObject)asyncResult.AsyncState;
78                  Socket worker = stateObject.workSocket;
79
80                  int byteInput = worker.EndReceive(asyncResult);
81
82                  if (byteInput > 0)
83                  {
84                      stateObject.stringBuilder.Append(encoding.GetString
                        (stateObject.buffer, 0, byteInput));
85
86                      transaction = stateObject.stringBuilder.ToString();
87
88                      if (transaction.IndexOf(TransactionEndTag) > -1)
89                      {
90                          Log($"Read {transaction.Length} bytes from socket. \n
                        Data : {transaction}");
91                          ProcessTransaction(worker, transaction);
92                      }
93                      else
94                      {
95                          worker.BeginReceive(stateObject.buffer, 0,
                        StateObject.BufferSize, 0, new AsyncCallback(ReadCallback),
```

```
                        stateObject);
 96                     }
 97                 }
 98             }
 99             catch
100             {
101
102             }
103         }
104
105         private void ProcessTransaction(Socket worker, string data)
106         {
107             MatchCollection matchCollection = transactionRegex.Matches(data);
108
109             if (matchCollection.Count != 0)
110             {
111                 GroupCollection groupCollection = matchCollection[0].Groups;
112
113                 string type = groupCollection[1].Value;
114                 string key = groupCollection[2].Value;
115                 string token = groupCollection[3].Value;
116                 Algorithm algorithm = ConvertStringToAlgorithm(groupCollection
                      [4].Value);
117                 string content = groupCollection[5].Value;
118
119                 data = Transaction(type, key, token, algorithm, content);
120             }
121             else
122             {
123                 data = InvalidTransaction("Incorrect Transaction Formatting");
124             }
125
126             Send(worker, data);
127         }
128
129         private void Send(Socket worker, string data)
130         {
131             byte[] byteArray = encoding.GetBytes(data);
132             worker.BeginSend(byteArray, 0, byteArray.Length, 0, new AsyncCallback
                  (SendCallback), worker);
133         }
134
135         private void SendCallback(IAsyncResult ayncResult)
136         {
137             try
138             {
139                 Socket worker = (Socket)ayncResult.AsyncState;
140                 int byteCount = worker.EndSend(ayncResult);
141                 Console.WriteLine($"Sent {byteCount} bytes to client.");
142                 worker.Shutdown(SocketShutdown.Both);
143                 worker.Close();
144             }
```

```
145             catch
146             {
147
148             }
149         }
150
151         private string Transaction(string type, string publicKey, string token,
              Algorithm algorithm, string content)
152         {
153             if (type.ToLower() == "exchange")
154             {
155                 content = Exchange(algorithm, publicKey);
156             }
157             else if (type.ToLower() == "createaccount")
158             {
159                 content = CreateAccount(algorithm, publicKey, token, content);
160             }
161             else if (type.ToLower() == "authenticate")
162             {
163                 content = Authenicate(algorithm, publicKey, token, content);
164             }
165             else if (type.ToLower() == "message")
166             {
167                 content = Message(algorithm, publicKey, token, content);
168             }
169             else if (type.ToLower().Contains("sendfile"))
170             {
171                 content = SendFile(algorithm, type, publicKey, token, content);
172             }
173             else if (type.ToLower() == "requestfile")
174             {
175                 content = RequestFile(algorithm, publicKey, token, content);
176             }
177             else
178             {
179                 content = InvalidTransaction("Incorrect Transaction Type");
180             }
181
182             return content;
183         }
184
185         private string Exchange(Algorithm algorithm, string key)
186         {
187             string serverPublicKey;
188
189             using (ECDiffieHellmanCng server = new ECDiffieHellmanCng())
190             {
191                 server.KeyDerivationFunction =
                      ECDiffieHellmanKeyDerivationFunction.Hash;
192                 server.HashAlgorithm = CngAlgorithm.Sha256;
193                 serverPublicKey = Convert.ToBase64String
                      (server.PublicKey.ToByteArray());
```

```
194                 byte[] clientPublicKey = Convert.FromBase64String(key);
195                 byte[] symKeyBytes = server.DeriveKeyMaterial(CngKey.Import
                      (clientPublicKey, CngKeyBlobFormat.EccPublicBlob));
196                 AddToRing(key, Convert.ToBase64String(symKeyBytes));
197             }
198
199             return TransactionFormat("exchange", key, serverPublicKey,
                  algorithm.ToString(), "");
200         }
201
202         private string CreateAccount(Algorithm algorithm, string publicKey,
              string token, string content)
203         {
204             string IV;
205
206             string symmetricKey = GetSymmetricKey(publicKey, Ring);
207
208             content = Decrypt(algorithm, symmetricKey, token, content);
209
210             MatchCollection matchCollection = createAccountRegex.Matches
                  (content);
211
212             if (matchCollection.Count != 0)
213             {
214                 GroupCollection groupCollection = matchCollection[0].Groups;
215
216                 string username = groupCollection[1].Value;
217                 string password = groupCollection[2].Value;
218
219                 if (username.Length >= 8)
220                 {
221                     if (IsUsername(username) == false)
222                     {
223                         if (AddAccount(username, password))
224                         {
225                             ProcessKey(publicKey);
226                             content = MessageFormat(algorithm, symmetricKey,
                          "CreatedAccount", out IV);
227                         }
228                         else
229                         {
230                             content = MessageFormat(algorithm, symmetricKey,
                          "FailedAccount", out IV);
231                         }
232                     }
233                     else
234                     {
235                         content = MessageFormat(algorithm, symmetricKey,
                          "InvalidUsernameExists", out IV);
236                     }
237                 }
238                 else
```

```
239                 {
240                     content = MessageFormat(algorithm, symmetricKey,
                        "InvalidUsernameLength", out IV);
241                 }
242             }
243             else
244             {
245                 content = MessageFormat(algorithm, symmetricKey,
                      "InvalidFormatCreateAccount", out IV);
246             }
247 
248             return TransactionFormat("createAccount", publicKey, IV,
                  algorithm.ToString(), content);
249         }
250 
251         private string Authenicate(Algorithm algorithm, string publicKey, string
              token, string content)
252         {
253             string IV;
254 
255             string symmetricKey = GetSymmetricKey(publicKey, Ring);
256 
257             content = Decrypt(algorithm, symmetricKey, token, content);
258 
259             MatchCollection matchCollection = authenicationRegex.Matches
                  (content);
260 
261             if (matchCollection.Count != 0)
262             {
263                 GroupCollection groupCollection = matchCollection[0].Groups;
264                 string username = groupCollection[1].Value;
265                 string password = groupCollection[2].Value;
266 
267                 if (IsUsername(username))
268                 {
269                     if (IsPassword(username, password))
270                     {
271                         ProcessKey(publicKey);
272                         content = MessageFormat(algorithm, symmetricKey,
                        "Authorized", out IV);
273                     }
274                     else
275                     {
276                         content = MessageFormat(algorithm, symmetricKey,
                        "InvalidPassword", out IV);
277                     }
278                 }
279                 else
280                 {
281                     content = MessageFormat(algorithm, symmetricKey,
                        "InvalidUsername", out IV);
282                 }
```

```
283             }
284             else
285             {
286                 content = MessageFormat(algorithm, symmetricKey,
                      "InvalidFormatAuthorize", out IV);
287             }
288
289             return TransactionFormat("authenication", publicKey, IV,
                  algorithm.ToString(), content);
290         }
291
292         private string Message(Algorithm algorithm, string publicKey, string
              token, string content)
293         {
294             string IV;
295
296             if (IsAuthorized(publicKey))
297             {
298                 string symmetricKey = GetSymmetricKey(publicKey, Ring);
299                 content = Decrypt(algorithm, symmetricKey, token, content);
300                 MatchCollection matchCollection = messageRegex.Matches(content);
301
302                 if (matchCollection.Count != 0)
303                 {
304                     GroupCollection groupCollection = matchCollection[0].Groups;
305                     string message = groupCollection[1].Value;
306                     Log($"Recieved '{message}' from client");
307                     content = MessageFormat(algorithm, symmetricKey,
                          "RecievedMessage", out IV);
308                 }
309                 else
310                 {
311                     Log($"Invalid Message Format Recieved {content}");
312                     content = MessageFormat(algorithm, symmetricKey,
                          "InvalidFormatMessage", out IV);
313                 }
314             }
315             else
316             {
317                 Log($"Unauthorized Request from {publicKey}");
318                 content = MessageFormat(algorithm, "", "NotAuthorized", out IV);
319             }
320
321             return TransactionFormat("message", publicKey, IV, algorithm.ToString
                  (), content);
322         }
323
324         private string SendFile(Algorithm algorithm, string type, string
              publicKey, string token, string content)
325         {
326             string IV;
327
```

```
328             if (IsAuthorized(publicKey))
329             {
330                 string symmetricKey = GetSymmetricKey(publicKey, Ring);
331                 content = Decrypt(algorithm, symmetricKey, token, content);
332                 MatchCollection matchCollection = messageRegex.Matches(content);
333 
334                 if (matchCollection.Count != 0)
335                 {
336                     GroupCollection groupCollection = matchCollection[0].Groups;
337                     string message = groupCollection[1].Value;
338                     string filetype = type.Substring(8);
339 
340                     Log($"Recieved file {filetype} from client");
341 
342                     SaveFile(filetype, message);
343                     content = MessageFormat(algorithm, symmetricKey,
                         $"RecievedFile{filetype}", out IV);
344                 }
345                 else
346                 {
347                     Log($"Invalid Message Format Recieved {content}");
348                     content = MessageFormat(algorithm, symmetricKey,
                         "InvalidFormatMessage", out IV);
349                 }
350             }
351             else
352             {
353                 Log($"Unauthorized Request from {publicKey}");
354                 content = MessageFormat(algorithm, "", "NotAuthorized", out IV);
355             }
356 
357             return TransactionFormat("message", publicKey, IV, algorithm.ToString
                  (), content);
358         }
359 
360         private string RequestFile(Algorithm algorithm, string publicKey, string
              token, string content)
361         {
362             string IV;
363 
364             if (IsAuthorized(publicKey))
365             {
366                 string symmetricKey = GetSymmetricKey(publicKey, Ring);
367                 content = Decrypt(algorithm, symmetricKey, token, content);
368                 MatchCollection matchCollection = messageRegex.Matches(content);
369 
370                 if (matchCollection.Count != 0)
371                 {
372                     GroupCollection groupCollection = matchCollection[0].Groups;
373                     string fileRequest = groupCollection[1].Value;
374                     Log($"Sent File {fileRequest} to client");
375                     content = MessageFormat(algorithm, symmetricKey, ReadFile
```

```
                    (fileRequest), out IV);
376                 }
377                 else
378                 {
379                     Log($"Invalid Message Format Recieved {content}");
380                     content = MessageFormat(algorithm, symmetricKey,
                        "InvalidFormatMessage", out IV);
381                 }
382             }
383             else
384             {
385                 Log($"Unauthorized Request from {publicKey}");
386                 content = MessageFormat(algorithm, "", "NotAuthorized", out IV);
387             }
388
389             return TransactionFormat("message", publicKey, IV, algorithm.ToString
                  (), content);
390         }
391
392         private string InvalidTransaction(string exception = "")
393         {
394             return TransactionFormat("exception", "", "", exception);
395         }
396
397         private string MessageFormat(Algorithm algorithm, string symmetricKey,
              string message, out string IV)
398         {
399             if (string.IsNullOrEmpty(symmetricKey))
400             {
401                 IV = string.Empty;
402                 return "{{Message = 'InvalidKeyDetected'}}";
403             }
404             else
405             {
406                 return Encrypt(algorithm, symmetricKey, "{{Message = '" + message
                       + "'}}", out IV);
407             }
408         }
409
410         private void AddToRing(string publicKey, string symmetricKey)
411         {
412             Ring.Add(new ClientObject(publicKey, symmetricKey));
413         }
414
415         private string GetSymmetricKey(string publicKey, List<ClientObject> Ring)
416         {
417             foreach (ClientObject client in Ring)
418             {
419                 if (client.PublicKey == publicKey)
420                 {
421                     return client.SymmetricKey;
422                 }
```

```
423             }
424             return string.Empty;
425         }
426
427         private string ProcessKey(string key)
428         {
429             bool truth = false;
430
431             foreach (ClientObject client in Ring)
432             {
433                 if (client.PublicKey == key)
434                 {
435                     truth = true;
436                     client.Authenticated = true;
437                     Log($"Successfully Authenticated Key: {client.PublicKey}");
438                     break;
439                 }
440             }
441
442             if (!truth)
443                 Log($"Warning unable to authenticate key. The ring was not
                      updated.");
444
445             return key;
446         }
447
448         private bool IsAuthorized(string key)
449         {
450             bool truth = false;
451
452             foreach (ClientObject client in Ring)
453             {
454                 if (client.PublicKey == key)
455                 {
456                     truth = true;
457                     break;
458                 }
459             }
460             return truth;
461         }
462
463         private void SaveFile(string filename, string data)
464         {
465
466             Files file = ConvertStringToFiles(filename);
467
468             if (file == Files.Sales)
469             {
470                 File.WriteAllText(@"..\..\Static\FilesIn\Sales.txt", data);
471             }
472             else if (file == Files.Maps)
473             {
```

```
474                     File.WriteAllText(@"..\..\Static\FilesIn\Maps.txt", data);
475                 }
476                 else if (file == Files.Budget)
477                 {
478                     File.WriteAllText(@"..\..\Static\FilesIn\Budget.txt", data);
479                 }
480                 else
481                 {
482                     File.WriteAllText(@"..\..\Static\FilesIn\Error.txt", data);
483                 }
484             }
485 
486             private string ReadFile(string filename)
487             {
488                 Files file = ConvertStringToFiles(filename);
489 
490                 string fileContent;
491                 if (file == Files.Sales)
492                 {
493                     fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                        \Sales.txt");
494                 }
495                 else if (file == Files.Maps)
496                 {
497                     fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                        \Maps.txt");
498                 }
499                 else if (file == Files.Budget)
500                 {
501                     fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                        \Budget.txt");
502                 }
503                 else
504                 {
505                     fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                        \Error.txt");
506                 }
507                 return fileContent;
508             }
509 
510 
511             private void Log(string text)
512             {
513                 _logBox.Dispatcher.Invoke(() =>
514                 {
515                     _logBox.Text = text;
516                 });
517                 Console.WriteLine(text);
518             }
519         }
520     }
521 
```