

# **BSc Computer Science for Games Programming**

## **Distributed Systems Programming**

### **Assessed Coursework Report**

**Name: Samuel Lawrence**

**Student Number: 202227714**

**Word Count: 1508**

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Design and Implementation</b>	<b>2</b>
Server Architecture and Stateless API Design	2
RESTful Client-Server Principles	3
Authentication and Authorization Strategy	3
Database Design and Logging	3
Encryption and Security Mechanisms	4
Testing and Debugging Approach	5
Reflective Statement	6
Tasks Completed and Level of Completion	6
Challenges Encountered	6
Solutions and Learning Outcomes	6
Conclusion	7
<b>References</b>	<b>8</b>
<b>Appendix</b>	<b>9</b>
Screenshots	9
GET /api/talkback/hello	9
GET /api/talkback/sort	10
GET api/user/new?username=TestUser	11
POST api/user/new?username=TestUser	12
DEL api/user/removeuser?username=TestUser	13
POST api/user/changerole	14
GET api/protected/hello	14
GET api/protected/sha1?message=hello	15
GET api/protected/sha256?message=hello	15
GET api/protected/getpublickey	16
GET /api/protected/sign?message=hello	17
GET /api/protected/mashify	18
GET api/other/clear	18

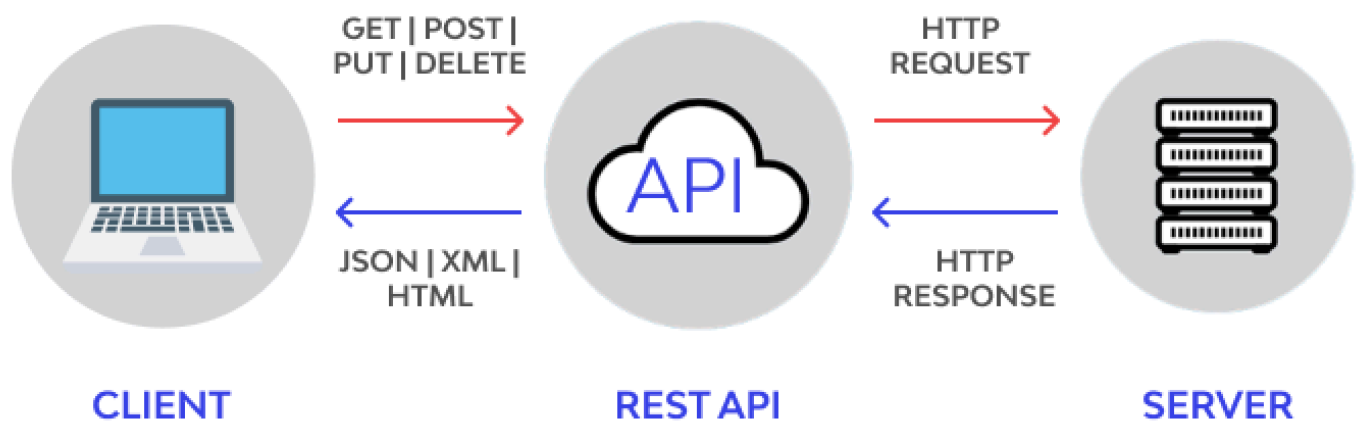
# Design and Implementation

The main goal for this project was to design, build, and fully test a reliable, scalable, and secure client-server system. I used ASP.NET Core Web API for the server-side and a C# console application for the client-side. We were given a basic project skeleton to start with, which I expanded the functionality on both sides significantly. The final system not only covers all the functional and security requirements outlined in the assignment brief, but also shows a strong focus on good design principles and practical security practices throughout.

## Server Architecture and Stateless API Design

From the beginning of the project, I wanted the server to follow a clean and fully stateless design. Every client request carries everything it needs, mainly by including an API Key inside the HTTP header. This meant the server didn't need to keep track of sessions or store user context between requests. Stateless designs like this are essential for cloud deployments where you might have multiple server instances behind a load balancer. If the server is stateless, it doesn't matter which instance handles a request.

To keep the code tidy and reusable, I structured all API controllers to inherit from a shared BaseController class. This class gave them easy access to common things like the database context and shared error-handling logic. Routing across the server was handled using attribute-based routes, following the simple `/api/[controller]/[action]` format. This kept endpoint paths clean, predictable, and easy to document. Security was layered on with attributes like `[Authorize]` to control access, keeping concerns separated properly and avoiding messy logic inside controllers themselves.



(Source: [Stateless-ness in RESTful APIs](#))

## RESTful Client-Server Principles

The server was designed to follow proper RESTful principles from top to bottom. I paid close attention to HTTP method usage:

- **GET** requests were used only for safe, read-only operations like fetching a hello message, sorting strings, requesting hashes, or retrieving public keys.
- **POST** requests were reserved for operations that modify server state, such as creating users, changing roles, or calling the Mashify feature.
- **DELETE** requests were used purely for deleting resources, like when removing a user account.

On top of that, HTTP status codes were carefully used across every endpoint. Successful operations returned 200 OK, client-side input problems resulted in 400 Bad Request, and forbidden access attempts triggered 403 Forbidden. I also put a lot of effort into centralized error handling through middleware. This meant controller methods stayed clean and easy to read, and any errors that did happen were caught and returned.

## Authentication and Authorisation Strategy

Security was a priority from the very start. I modified the authentication and authorization systems using two dedicated middleware layers: CustomAuthenticationHandlerMiddleware and CustomAuthorizationHandlerMiddleware. The authentication middleware checked that the client had a valid API Key included in the request, while the authorisation middleware enforced fine-grained access controls. This setup is very important for admin-only routes like Protected/Mashify.

## Database Design and Logging

For persistence, I used Entity Framework Core with a code-first approach. The main database tables were User, Log, and LogArchive. Special care was taken to make sure the user-log relationships were handled properly so that every action a user performed could be audited later if needed.

Whenever a user was deleted, their related logs weren't just wiped out. Instead, they were moved into a separate LogArchive table, keeping the audit trail intact without cluttering the active database tables. This archiving step ensured that historical user activity could always be reviewed even if the user account itself no longer existed.

All database access was handled through a dedicated UserRepository class, keeping database logic separated from the controller and service layers. This made it easier to maintain and test the system later on. Although the skeleton was set up for SQL Server, I got permission from John Dixon to use SQLite for development purposes, as long as the final submission stayed compatible with the original SQL Server setup.

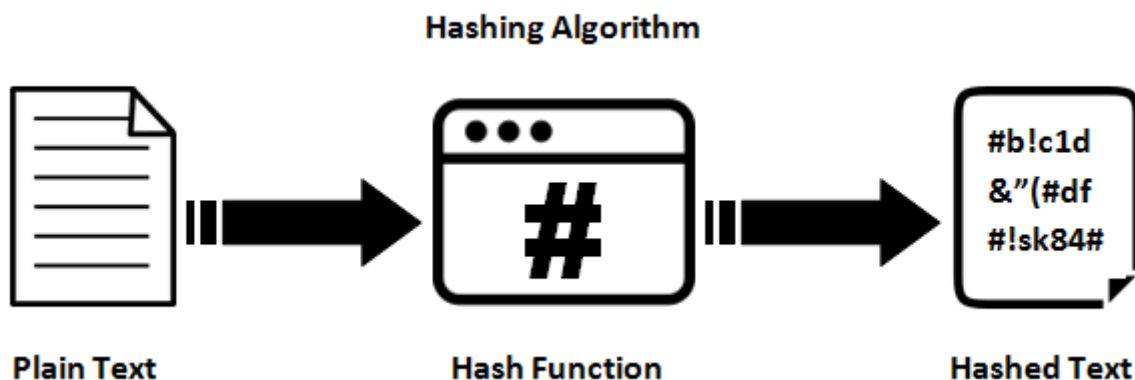
# Encryption and Security Mechanisms

Implementing the cryptographic requirements was one of the more technical parts of the project. I implemented several different security mechanisms:

- SHA1 and SHA256 hashing for generating secure message digests.
- RSA encryption using OAEP-SHA1 padding for public-key operations.
- AES encryption using CBC mode with PKCS7 padding for securely handling the Mashify feature.

One of the more challenging parts was making sure that encryption, decryption, and hashing worked exactly the same way as the official test server, down to small details like encoding and padding. In some cases, the slightest difference (like using UTF-8 versus ASCII at the wrong stage) would cause cryptographic operations to silently fail, leading to mismatched hashes or decryption errors which would not meet the specified requirements.

To avoid those issues, I tried to pay close attention to when ASCII encoding should be used (like when handling hex representations for digital signatures) and when UTF-8 was needed to properly support a broader range of characters.



(Source: [Breaking Down : SHA-256 Algorithm | by Aditya Anand | InfoSec Write-ups](#))

## Testing and Debugging Approach

On the client side, all network communication was built around `async/await` patterns to avoid freezing the console while waiting for server responses. Some specific debugging techniques that helped a lot included:

- Manually checking the presence and accuracy of API Key headers in requests.
- Inspecting raw byte arrays during encryption and decryption processes to catch subtle bugs in AES or RSA operations.
- Comparing computed hashes character-by-character against reference results to detect small data mismatches.
- Building a full Postman collection to rapidly retest all endpoints after any code change, catching regressions early.

One area that gave me more trouble than expected was the Mashify feature. Implementing AES encryption correctly turned out to be extremely difficult. Even small mistakes in the way keys or IVs were handled would completely break encryption or decryption, and the server would either reject the request silently or fail to decrypt properly. At one point, I spent hours just trying to figure out whether the problem was in the encoding of the data before encryption, the block size, the way padding was applied, or the transformation of byte arrays into hex strings. Debugging Mashify took longer than any other feature, and taught me just how fragile cryptographic implementations can be when even tiny assumptions go wrong.

Getting Mashify working in the end felt like a massive win, but it definitely exposed how easy it is to make subtle mistakes when dealing with encryption across distributed systems.

# Reflective Statement

## Tasks Completed and Level of Completion

- Fully developed the server to the required standard, implementing all required features.
- Built a functioning C# console client with asynchronous network communication.
- Integrated secure authentication, authorisation, and detailed logging.
- Successfully implemented RSA digital signatures and AES encryption for the Mashify feature.
- Thoroughly tested the server and client against the reference server, matching expected outputs consistently.

## Challenges Encountered

- Debugging RSA signature verification was significantly more complex and brittle than expected. Tiny mismatches between byte arrays and hex string representations caused a lot of early failures.
- Handling Entity Framework migrations without accidentally losing data when tweaking the schema mid-project.
- Diagnosing issues with AES decryption, where small differences in keys or IVs would cause operations to fail.
- Making sure encrypted payloads were formatted exactly as expected by the test server, including things like dash-separated hex encoding.

## Solutions and Learning Outcomes

One of the biggest lessons from this project was the importance of small implementation details. In distributed systems, tiny problems like an encoding mismatch or a missing dash in a payload could break communication completely. Learning how to systematically track down and fix these problems made me much more confident working at both the client and server levels. I also hope to use what I have learned here for my honours stage project.

Incremental testing and validation after every task saved huge amounts of time later. Being able to quickly spot where issues were happening meant that problems were fixed while they were still small, manageable and fresh in my head.

## Conclusion

Finishing this project was one of the most valuable technical experiences I've had. It taught me a wide range of skills, secure API development, cryptography, database management, and distributed systems debugging, and tied them together inside a real-world architecture.

One of the biggest takeaways was how critical attention to detail becomes when building secure systems. Seemingly small mistakes in encryption or lazy implementation with request formatting, or API design can cause entire systems to break down, and being able to spot and fix those issues is a major skill I gained.

This project also reinforced how important it is to design with future scalability and security in mind, not just to meet minimum requirements. Thinking about how to better manage user data and how to harden endpoints against abuse made the whole system stronger and closer to what you'd need in real-world scenarios.

Overall, I'm proud of what I built, not just for its functionality but for the design thinking behind it. This project pushed my technical skills forward and gave me a much stronger foundation for tackling even bigger distributed system projects in the future.



# References

- Microsoft. Entity Framework Core Documentation. Microsoft Learn. Available at: <https://learn.microsoft.com/en-us/ef/core/> [Accessed 12 Mar. 2025].
- Microsoft. ASP.NET Core Web API. Microsoft Learn. Available at: <https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0> [Accessed 15 Mar. 2025].
- Microsoft. Authentication and Authorization in ASP.NET Core. Microsoft Learn. Available at: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/> [Accessed 20 Mar. 2025].
- Open Web Application Security Project (OWASP). API Security Top 10. OWASP. Available at: <https://owasp.org/www-project-api-security/> [Accessed 18 Mar. 2025].
- Microsoft. Cryptography Model (C#). Microsoft Learn. Available at: <https://learn.microsoft.com/en-us/dotnet/standard/security/cryptography-model> [Accessed 25 Mar. 2025].
- Kalema Edgar. Stateless-ness in RESTful APIs. Medium. Available at: <https://kalemaedgar.medium.com/stateless-ness-in-restful-apis-65db25dc96a1> [Accessed 10 Mar. 2025].
- InfoSec Writeups. Breaking Down SHA-256 Algorithm. Available at: <https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3> [Accessed 10 Mar. 2025].

# Appendix

## Screenshots

GET /api/talkback/hello

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** https://localhost:44394/api/talkback/hello
- Buttons:** Send, Params, Auth, Headers (7), Body, Scripts, Settings, Cookies
- Headers:** 6 hidden. Visible header: Content-Type: application/json
- Body:** 200 OK, 1.36 s, 352 B. Raw view shows: 1 Hello World

```
...please wait...
Hello World

What would you like to do next?
```

## GET /api/talkback/sort

GET <https://localhost:44394/api/talkback/sort?integers=6&i...> Send

Params • Auth Headers (7) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Descrip...	Bulk Edit
<input checked="" type="checkbox"/>	integers	6		
<input checked="" type="checkbox"/>	integers	1		

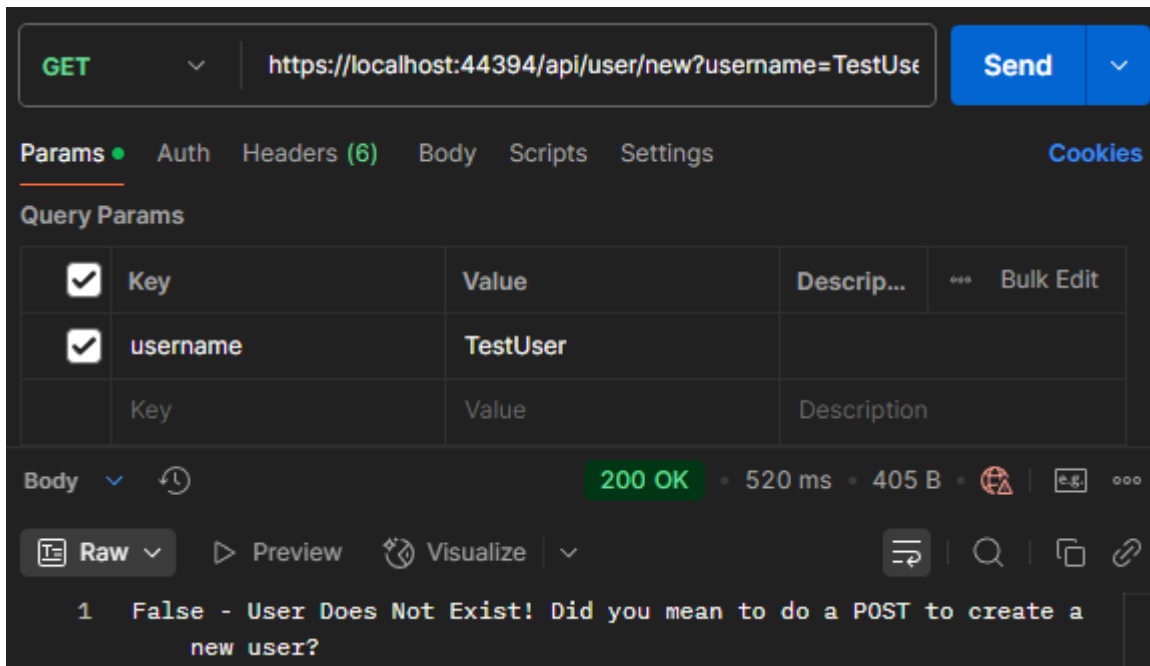
Body ☒ ☐ ☐ 200 OK • 37 ms • 193 B •

☒ JSON ☐ Preview ☐ Visualize ☐

```
1  [  
2    1,  
3    3,  
4    4,  
5    6,  
6    8  
7  ]
```

```
...please wait...  
[1,3,4,6,8]  
What would you like to do next?
```

GET api/user/new?username=TestUser



GET <https://localhost:44394/api/user/new?username=TestUser> Send

Params • Auth Headers (6) Body Scripts Settings Cookies

Query Params

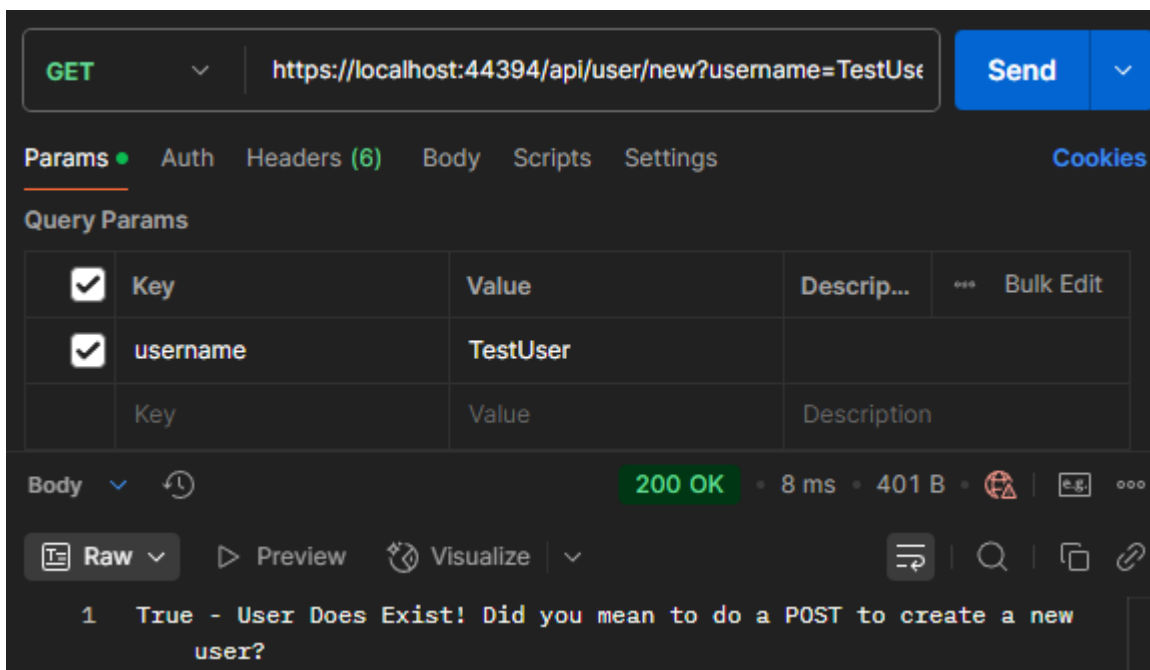
<input checked="" type="checkbox"/>	Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/>	username	TestUser		
	Key	Value	Description	

Body ☒ 200 OK • 520 ms • 405 B • [Error Icon] [Copy Icon] [More Icon]

Raw ☒ Preview Visualize ☒

```
1 False - User Does Not Exist! Did you mean to do a POST to create a new user?
```

```
...please wait...
False - User Does Not Exist! Did you mean to do a POST to create a new user?
What would you like to do next?
```



GET <https://localhost:44394/api/user/new?username=TestUser> Send

Params • Auth Headers (6) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/>	username	TestUser		
	Key	Value	Description	

Body ☒ 200 OK • 8 ms • 401 B • [Error Icon] [Copy Icon] [More Icon]

Raw ☒ Preview Visualize ☒

```
1 True - User Does Exist! Did you mean to do a POST to create a new user?
```

```
...please wait...
True - User Does Exist! Did you mean to do a POST to create a new user?
What would you like to do next?
```

## POST api/user/new?username=TestUser

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** https://localhost:44394/api/user/new
- Body:** "TestUser"
- Response:** 200 OK, 11 ms, 379 B
- Raw Response:** 3596b9fb-308f-46df-a212-a779d714fcf5

```
...please wait...  
Got API Key  
What would you like to do next?
```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** https://localhost:44394/api/user/new
- Body:** "TestUser"
- Response:** 403 Forbidden, 43 ms, 259 B
- Raw Response:** Oops. This username is already in use. Please try again with a new username.

```
...please wait...  
Error: Oops. This username is already in use. Please try again with a new username.  
What would you like to do next?
```

DEL api/user/removeuser?username=TestUser

**DELETE** ▼ <https://localhost:44394/api/User/RemoveUser?username=TestUser> **Send** ▼

Params ● Auth **Headers (8)** Body Scripts Settings Cookies

	Key	Value	...	Bulk Edit	Presets ▼
<input type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	ApiKey	83a595b2-8158-4be5-9...			
	Key	Value			Description

Body ▼ 🔄 **200 OK** • 46 ms • 186 B • 🌐 | 📄 ...

{} **JSON** ▼ ▶ Preview 🔄 Visualize ▼ ↺ ≡ 🔍 📄 🔗

1 `true`

```
...please wait...
True
What would you like to do next?
```

**DELETE** ▼ <https://localhost:44394/api/User/RemoveUser?username=TestUser> **Send** ▼

Params ● Auth **Headers (8)** Body Scripts Settings Cookies

	Key	Value	...	Bulk Edit	Presets ▼
<input type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	ApiKey	83a595b2-8158-4be5-9...			
	Key	Value			Description

Body ▼ 🔄 **200 OK** • 10 ms • 187 B • 🌐 | 📄 ...

{} **JSON** ▼ ▶ Preview 🔄 Visualize ▼ ↺ ≡ 🔍 📄 🔗

1 `false`

## POST api/user/changerole

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** https://localhost:44394/api/user/changerole
- Body:** JSON format with the following content:

```
{  "username" : "TestUser2",  "role" : "Admin"}
```
- Status:** 200 OK
- Response Time:** 30 ms
- Response Size:** 347 B
- Response Headers:** Content-Type: application/json
- Response Body:** Raw view showing the JSON response.

```
...please wait...
DONE
What would you like to do next?
```

## GET api/protected/hello

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** https://localhost:44394/api/protected/hello
- Headers:** 8 headers are listed, including Content-Type (application/json) and ApiKey (06255ba1-1c9d-41b1-95...).
- Status:** 200 OK
- Response Time:** 147 ms
- Response Size:** 354 B
- Response Headers:** Content-Type: application/json
- Response Body:** Raw view showing the response: Hello TestUser

```
...please wait...
Hello TestUser
What would you like to do next?
```

GET api/protected/sha1?message=hello

GET <https://localhost:44394/api/protected/sha1?message=hello> Send

Params • Auth Headers (8) Body Scripts Settings Cookies

	Key	Value	
<input checked="" type="checkbox"/>	Content-Type	application/json	
<input checked="" type="checkbox"/>	ApiKey	06255ba1-1c9d-41b1-95...	
	Key	Value	Description

Body 200 OK • 13 ms • 384 B

Raw Preview Visualize

```
1 AAF4C61DDCC5E8A2DABEDE0F3B482CD9AEA9434D
```

```
...please wait...
AAF4C61DDCC5E8A2DABEDE0F3B482CD9AEA9434D
What would you like to do next?
```

GET api/protected/sha256?message=hello

GET <https://localhost:44394/api/protected/sha256?message=hello> Send

Params • Auth Headers (8) Body Scripts Settings Cookies

	Key	Value	
<input checked="" type="checkbox"/>	Content-Type	application/json	
<input checked="" type="checkbox"/>	ApiKey	06255ba1-1c9d-41b1-95...	
	Key	Value	Description

Body 200 OK • 64 ms • 410 B

Raw Preview Visualize

```
1 2CF24DBA5FB0A30E26E83B2AC5B9E29E1B161E5C1FA7425E73043362938B9824
```

```
...please wait...
2CF24DBA5FB0A30E26E83B2AC5B9E29E1B161E5C1FA7425E73043362938B9824
What would you like to do next?
```



## GET api/protected/getpublickey

GET <https://localhost:44394/api/protected/getpublickey> Send

Params Auth Headers (8) Body Scripts Settings Cookies

	Key	Value	
<input checked="" type="checkbox"/>	Content-Type	application/json	
<input checked="" type="checkbox"/>	ApiKey	06255ba1-1c9d-41b1-95...	
	Key	Value	Description

Body 200 OK • 12 ms • 581 B •

**Raw** Preview Visualize

```
1 <RSAKeyValue><Modulus>nv0VAQ2l4Eqp
+n9EqS9Ea00XitgYgbabaACbNvItxE8joRoAQjMWpWZNzUYLYu+iSGEB//
nBPWII6fLDCLDIItsQt9VGgTpyPEpp9lwBX1FinzYrE4Uc5wLt7tSVgWgP7BXYQ9V
gW7wvM9uJS8fqhYGPpyFTXjIENKLxui+RcW0vE=</Modulus><Exponent>AQAB</
Exponent></RSAKeyValue>
```

...please wait...

Got Public Key

What would you like to do next?

GET /api/protected/sign?message=hello

GET https://localhost:44394/api/protected/sign?message=h... Send

Params Auth Headers (8) Body Scripts Settings Cookies

	Key	Value	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	ApiKey	06255ba1-1c9d-41b1-95...			
	Key	Value	Description		

Body 200 OK 36 ms 678 B e.g. ...

Raw Preview Visualize

```
1 65-7C-99-07-F2-0E-DD-8F-B3-19-B6-BA-76-92-D7-D1-6C-C2-9F-9E-F1-07-AC
  -17-B1-2D-1F-7D-D7-F8-7D-EE-3A-AA-0C-B3-86-F7-F7-62-E2-3B-78-50-
  C6-9B-7C-75-17-55-4E-4C-40-C2-69-75-C7-0A-FD-00-E8-8C-A0-76-EC-6
  D-97-B3-64-5F-8B-BC-F2-1C-0D-32-78-79-6A-67-36-29-E6-80-04-BA-C3
  -84-C1-B8-99-10-49-6A-32-77-A7-6F-13-50-F4-29-95-B5-4C-24-7E-69-
  86-DE-EB-1F-5E-16-63-3F-D8-0F-3B-0B-6D-59-D5-3F-0B-5C-35-99
```

```
...please wait...
Message was successfully signed
What would you like to do next?
```

## GET /api/protected/mashify

GET <https://localhost:44394/api/protected/mashify> Send

Params Auth Headers (9) **Body** Scripts Settings Cookies

raw JSON Beautify

```
1 {  
2   "EncryptedString":  
   "69-BA-18-E5-08-DF-FB-01-9B-53-30-32-30-D3-69-AE-1E-0D-DC-0A-C  
   6-C7-F7-BB-AD-BA-C4-DF-BF-B7-44-18-99-D6-A8-5E-F2-9B-8C-F8-E1-
```

Body 200 OK • 4.07 s • 392 B

Raw Preview Visualize

```
1 69-46-E4-0C-5C-CF-ED-11-32-1C-60-FB-2C-E7-5F-93
```

```
...please wait...  
tsXT  
  
What would you like to do next?
```

## GET api/other/clear

GET <https://localhost:44394/api/other/clear> Send

Params Auth Headers (6) **Body** Scripts Settings Cookies

none

This request does not have a body

Body 200 OK • 38 ms • 363 B

Raw Preview Visualize

```
1 Success, all data cleared.
```