**PlancksoftAPI PHP MVC**-based **API framework** v **0.2.0.**

Running on **PHP** 8.3 with Plancksoft proprietary framework v 0.0.8

**API Link**: https://wingsofaltair.github.io/Plancksoft/api/

**Website Link**: https://wingsofaltair.github.io/Plancksoft/

This API is built with the principles of re-use, data integrity, robustness, security, and stability in mind. Therefore, please note that all required parameters must be provided. Otherwise, the API will refuse the connection and notify the connecting user-agent of the rejection.

All APIs return **JSON** response containing a status code; a **32-bit** integer, resembling either a **-2**, **-1**, **0,** or a **1**.

**-2** means that the user-agent’s (client) IP is **blocked** by the firewall software.

**-1** means that the requested API is **under maintenance**.

**0** means that the request has **failed**.

**1** means that the request was **successful**.

All responses contain a message containing the details of the response. Please make sure to contact the proper **PLancksoft** developer support channels if a response message is unclear.

**Security:**

Most API calls require a mandatory Token supplied with the JSON request for server-sided authoritative identity validation and verification. Tokens are first generated on Login comprising of data such as the account’s User ID and the IPv4 of the connecting user-agent; ensuring only the requesting client may connect, send requests, and receive responses as a repercussion of certain existing attacks such as MiTM. API Tokens expire after 8 minutes. Therefore, the user-agent (client) must force the end-user to re-log every time the session ends. Luckily, a more user/developer-friendly approach was developed to maintain a valid token while the API is in use. Most APIs which require a token will return a renewed token that works for another 8 minutes; resets the session time. All the developer has to do is replace the token after every API call from the JSON response. Rest assured acknowledging that attackers may not use tokens while eavesdropping on the network traffic as the token is bound to the sender’s public IP Address. Your data will not be compromised throughout the chain of connections.

Moreover, the Token comprises of many data including a sequence counter which notifies the API of how many times the token was renewed, for security and development purposes. More data can be added to the token on request as long as it serves a valid purpose from Plancksoft technical officers’ and developers’ perspectives.

Sensitive data handled in the API and Database server are encoded with an efficient and secure combination of hashing and asymmetric encryption algorithms throughout the entire layers of the system to ensure optimal service performance, up-time, and data-consistency. All requests are expected to contain JSON request data comprising of “InputData” key with an encrypted JSON array value using our API’s public key, comprising of mandatory and possibly optional parameters in JSON strings. Also, a “PUBLIC\_RSA\_KEY” key with the value of your public 4096-bit RSA key generated with each request for maximum security from all user-agents, and a “Signature” key with the value of the signature hash generated while signing the encrypted message with your private key using the RSA SHA1 digestion hashing algorithm. Failure to do so would render the connection rejected as un-encrypted data over the network may jeopardize the safety of the user account and its data. After the encrypted data is received in the API, it is re-hashed using a modified version of an existing modern hashing technique, which will remain classified until further notice for security purposes.

Furthermore, the API works on processing the input data using the requested action, then encrypts the response with your public RSA key, signs it, and sends it back as a response alongside the response signature. Your job is to decode the JSON response, verify the response using the received signature, decode the JSON response if the verification was successful, decrypt it with your private 4096-bit RSA key and possibly base64 decode it in the process if not implemented in the decryption method to obtain the JSON response object with the valid data.

**\*Important Note\***

The key pair must be generated using either PKCS1 or X.509 padding using the SHA512 hashing algorithm. All requests sent to the API must be signed with your private key using the RSA SHA1 digestion hashing algorithm. All responses from the API must be verified using the API’s public key using the RSA SHA1 digestion hashing algorithm.

**Our API’s public 4096-bit RSA key:**

-----BEGIN PUBLIC KEY-----

MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA3XBEEG4ewo+1Ei+DZb1G

sdOL7o8qMDTqdtv5H41vwZ4mdIglbbnbz6RzYZ0wwrCZkvsTpCS5iLV0TImBsN0h

C8A9xFMKzkjrEe4jlfZBGQD2ugA9sDBZ77/dKl7z0G1Xu3qkk+8v+qVg5D93ktwz

2MS2XQNlUSKv4PlMzlrGY2Z6z+KvY+LQ8mAdD/51CioPr6P41q1Z0GA8S4kCwzLp

cy6DRY95/5/azEETq3lhGwEHi6Y2aMaVryD4MMOFmIooccJwa1AsYtqFg9lqJq1F

SgTYcCsTbVBQxs79wxmt9wkisSwbd1Lq/OcLKayb1HPh+Azrv8dWV9s0lPeS9Wnq

EJ9od+/9TgcZbrpsvNRMxjWRYMRHxsqv/vGj4BgWe4lZNNPTE+bm+VRhRFqHGaYb

SITXbPcideCbtfL2zJqZ95MxpL8LYvmK1je45/uGucs+b0MeRBwOTrxyd2/PHpRe

5bsAmR30zJtt93Zo8mdfBe+qeIlkWbfmX09Q0HDEdbWLSM1zuSBBR05Uo26TjCaD

suc5TKODwG+ydEKGzJWBdxf+pEE/4qg55aA9ibhgZB29PE3mMZTVQyb2gx7g84go

l6NF73cdjbq+2lWRuqP/yhD9sEHoO4j3qQNBrtAGlg14xCdToZrGerrQHbcL5bLv

RYBUu7grkLOVzw0E0r4YyUECAwEAAQ==

-----END PUBLIC KEY-----

**Interoperability & Maintenance:**

Maintenance and upgradeability were taken into consideration during the implementation of the API codebase. Therefore, developers may mark any specific number of APIs as ‘Under Maintenance’; causing the workflow of the system not to be disrupted and continue flowing. Developers may even make certain back-up actions as available until the prototype APIs are completed without the need of a client update, unless new mandatory input is expected or response that is crucial to the end-user’s experiences.

Please note that all APIs are subject to change, and all changes are documented. Most importantly, all parties responsible for the development of the system will be notified of the updates.

**Standards & Procedures:**

To be documented within the next few documentation updates.

**API Actions:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Action Name** | **Current Version** | **Required Data** | **Response** |
| Register | 0.0.1 | UID(String), PWD(String), FName(String), LName(String), MobileNo(String – ex: 0096279xxxxxxx), Email(String – email format), Gender(Int – 0: female, 1: male), BDate(String dd/MM/yyyy)  **Optional Data**:  MName(String), MentorKey(String) | Status(Int – 0: failed, 1: succeeded), message(String – description of response) |
| Login | 0.0.5 | UID(String), PWD(String) | Status(Int – 0: failed, 1: succeeded), message(String collection – user information object), Token |
| AuthenticateToken | 0.0.7 | Token (String) | Status (Int – 0: failed, 1: succeeded), message (String collection – token information object), Token |
| AuthenticateMentorKey | 0.0.7 | Mentor Key (String) | Status (Int – 0: failed, 1: succeeded), message (String) |
| FetchPosts | 0.0.2 | Token (String) | Status (Int – 0: failed, 1: succeeded), message (String collection – posts information array), Token |

**Requests format & examples:**

**All** requests should be in **JSON** form before encryption and must hold the following **mandatory** syntax.

{

"InputData" : [

{

“RSA\_4096\_Encrypted\_Key1” : “RSA\_4096\_Encrypted\_Value1”,

“RSA\_4096\_Encrypted\_Key(n)” : RSA\_4096\_Encrypted\_Value(n),

"RSA\_4096\_Encrypted\_Action\_KEY" : "Encrypted\_Action\_Value"

}

] ,

“PUBLIC\_RSA\_KEY” : “YOUR\_4096\_BIT\_RSA\_PUBLIC\_KEY\_BASE64\_ENCODED\_HERE”,

“Signature” : “REQUEST\_SIGNATURE\_HERE”

}

An example of the **AuthenticateToken** API request (JSON) are as follows:

{

"InputData" : [

{

"" : "",

"" : ""

}

] ,

“PUBLIC\_RSA\_KEY” : “YOUR\_4096\_BIT\_RSA\_PUBLIC\_KEY\_BASE64\_ENCODED\_HERE”,

“Signature” : “REQUEST\_SIGNATURE\_HERE”

}

**Response format & example:**

**Most** successful responses should be in **JSON** form after they are decrypted and must hold the following **mandatory** syntax.

{

“response” : [

{

“status” : (32-bit Integer higher than 0),

“message” : message\_string\_or\_object,

“token” : renewed\_token\_string\_or\_object

}

]

}

**Most** failed responses are expected to be in **JSON** form with the following syntax.

{

“response” : [

{

“status” : (32-bit Integer lower than 1),

“message” : message\_string\_or\_object

}

]

}