

# API

This document provides an overview and documentation for the API implemented in the provided code. The API allows clients to interact with a server for managing agents and uploading files.

- [Base URL](#)
- [Endpoints](#)
- [Database Connection](#)
- [User Folder Creation](#)
- [RSA Key Pair Generation and Storage](#)

# Base URL

The API create its own nodejs server at this address

`http://localhost:5000/`

# Endpoints

## Create a New Agent

- **URL:** `/api/agent/new`
- **Method:** **POST**
- **Description:** Create a new agent and store its information in a MySQL database.
- **Request Body:**
  - **versionOS** (required): The version of the operating system running on the agent.
  - **host** (required): The host name or IP address of the agent.
  - **hookUser** (required): The hook user of the agent.
- **Response:**
  - Status Code: **200 OK** on success, **400 Bad Request** if parameters are missing or invalid, **500 Internal Server Error** if an error occurs during database insertion.
  - Response Body:

```
{
  "data": {
    "id": "<agentId>",
    "publicKey": "<publicKey>"
  },
  "error": {}
}
```

- **Example:**

```
POST /api/agent/new HTTP/1.1
Host: localhost:5000
Content-Type: application/json
```

```
{
  "versionOS": "1.0",
  "host": "example.com",
  "hookUser": "john.doe"
}
```

## Upload a File

- **URL:** `/api/file/upload/:user_folder`

- **Method: POST**
- **Description:** Upload a file and save it in the specified user folder.
- **Request Parameters:**
  - **user\_folder** (required): The user folder to which the file should be uploaded.
- **Request Body:**
  - The file to be uploaded should be sent as **multipart/form-data** with the file field name set to **file**.
- **Response:**
  - Status Code: **200 OK** on success, **400 Bad Request** if no file is uploaded.
  - Response Body:

```
{
  "data": {
    "success": "<filename> uploaded successfully."
  },
  "error": {}
}
```

- **Example:**

```
POST /api/file/upload/123 HTTP/1.1
Host: localhost:5000
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="example.txt"
Content-Type: text/plain

This is the content of the file.

-----WebKitFormBoundary7MA4YWxkTrZu0gW--
```

# Database Connection

The API connects to a MySQL database for storing agent information. The database connection details are specified using environment variables:

- **DB\_HOST:** The host name or IP address of the MySQL database.
- **DB\_USER:** The username for accessing the MySQL database.
- **DB\_PASS:** The password for accessing the MySQL database.
- **DB\_MAIN:** The name of the main database.

# User Folder Creation

The API creates a folder named `/CTHULHU/users` if it doesn't already exist. Additionally, for each agent created, a user-specific folder is created within `/CTHULHU/users`.

## Folder Structure

The folder structure is as follows:

- `/CTHULHU`
  - `/users`
    - `/<agentId_1>`
    - `/<agentId_2>`
    - ...
    - `/<agentId_n>`

## Folder Creation Process

When a new agent is created, the API performs the following steps:

1. Checks if the `/CTHULHU/users` folder exists.
2. If the folder doesn't exist, it is created.
3. Creates a user-specific folder within `/CTHULHU/users` for the agent using the agent's ID.

## Example

Let's assume that the API receives a request to create a new agent with the ID `123`. Here's the folder creation process:

1. The API checks if the `/CTHULHU/users` folder exists.
2. If the folder doesn't exist, it creates the `/CTHULHU/users` folder.
3. The API creates a user-specific folder named `/CTHULHU/users/123` for the agent with ID `123`.

The created folder structure would be:

- `/CTHULHU`
  - `/users`
    - `/123`

# RSA Key Pair Generation and Storage

For each agent created, the API generates an RSA key pair consisting of a public key and a private key. The key pair is generated using a modulus length of 4096 bits. The generated keys are stored in the MySQL database along with other agent information.

## Key Generation Process

When a new agent is created, the API performs the following steps to generate the RSA key pair:

1. Generates an RSA key pair using the `crypto.generateKeyPairSync` method with the following options:
  - Algorithm: RSA
  - Modulus Length: 4096 bits
  - Public Key Encoding: PKCS#1 format in PEM encoding
  - Private Key Encoding: PKCS#1 format in PEM encoding
2. The generated public key and private key are converted to string representations.
3. The public key string is stored in the `pubKey` field of the agent's record in the MySQL database.
4. The private key string is stored in the `privKey` field of the agent's record in the MySQL database.

## Example

When a new agent is created, the API generates an RSA key pair. Let's assume the key generation process produces the following keys:

- Public Key:

```
-----BEGIN RSA PUBLIC KEY-----
MIICljANBgkqhkiG9w0BAQEFAAOCAg8AMIICGgKCAgEAYDRa5PEVYI2T3EzvG1on
...
d8Ou9azXQIDAQAB
-----END RSA PUBLIC KEY-----
```

- Private Key:

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIJKQIBAAKCAgEAYDRa5PEVYI2T3EzvG1onC4vPwL...
```

```
...
```

```
8RTgj8SPaHv/SmB2DhYO98C6HpU=
```

```
-----END RSA PRIVATE KEY-----
```

The generated keys are then stored in the agent's record in the MySQL database.