A Complete Guide to Developing a NodeJS Library for the Ksher Payment Gateway

Part 1: Architectural Overview of the Ksher Payment Gateway for Developers

This section provides the essential business and technical background for developers to understand the context of Ksher and effectively strategize their system integration. It will focus on information directly relevant to library development.

1.1. Introduction to Ksher as a Fintech Solution

Ksher is a provider of cross-border payment solutions, founded in 2016 and operating in over 9 countries worldwide, with a trusted client base of more than 200,000.[1] In Thailand, Ksher operates under the name "Ksher Payment Co., Ltd." and is licensed for payment services under the supervision of the Ministry of Finance, ensuring high standards of security and reliability.[5] The Ksher platform is designed as an all-in-one payment system that consolidates both offline and online payments to meet the needs of businesses of all sizes, from SMEs to large corporations.[6]

What makes Ksher stand out and gives it a strategic edge is its focus on cross-border commerce, particularly in facilitating payments for businesses in Thailand from Chinese tourists—a large market with high purchasing power. This is evident in Ksher's documentation and communications, which often emphasize support for popular Chinese payment channels like Alipay and WeChat Pay.[8] A clear case study is the partnership with Merlin Entertainments (Thailand) Ltd., the operator of SEA LIFE Bangkok Ocean World and Madame Tussauds Bangkok, which are popular tourist destinations for the Chinese.[8] This collaboration was not

just about adding payment channels but offering a solution that directly addresses the cashless behavior of Chinese tourists, capable of displaying exchange rates in Chinese Yuan instantly at the point of sale.[8]

For developers, understanding this strategic positioning is crucial. It indicates that Ksher's API is not designed merely as a generic payment gateway but has an architecture that supports cross-currency transactions, real-time exchange rate display, and integration with specialized platforms like WeChat Mini-Programs.[8] Therefore, the development of a NodeJS library should consider these functions, possibly by creating helper functions for currency management or preparing appropriate data structures (payloads) specifically for payment channels like Alipay and WeChat Pay, allowing library users to fully leverage Ksher's capabilities.

1.2. Supported Payment Channels and Usage Models

Ksher supports a wide and comprehensive range of payment channels to meet the demands of the digital age, including:

- **E-Wallets:** Supports leading domestic and international e-wallets such as Alipay, WeChat Pay, PromptPay, TrueMoney Wallet, Line Pay, and AirPay.[9]
- **Credit/Debit Cards:** Supports cards from major global networks including Visa, Mastercard, JCB, and UnionPay.[9]
- **Bank Transfers:** Supports payments via Mobile Banking Applications of leading banks.[13]

In addition to the variety of channels, Ksher offers solutions tailored to different business models:

- **E-commerce:** Integration via API/SDK for websites and applications.[14]
- **Social Commerce:** The LinkPay service allows merchants to generate payment links and send them to customers via social media or chat channels without needing a website.[14]
- **Offline/In-Store:** Payment acceptance through QR Code scanning and Electronic Data Capture (EDC) terminals.[8]

For developers aiming to build a library, understanding Ksher's API architecture is paramount. The API is not a single endpoint for "creating a payment" but is divided into distinct **Transaction Flows** to support various user experiences.[11] The main flows developers need to be aware of are:

1. **Redirect API (Gateway Pay):** This flow is ideal for e-commerce websites. The system generates a payment link and redirects the user to Ksher's payment page or the e-wallet provider's page directly. Upon completion, the user is redirected back to the merchant's website.[13]

2. **C-Scan-B API (Native Pay):** Short for "Customer Scans Business," this flow involves the merchant generating a dynamic QR Code on a screen (website, Kiosk), which the customer then scans with their e-wallet app to pay.[16]
3. **B-Scan-C API (Quick Pay):** Short for "Business Scans Customer," this flow is used at the point of sale. The customer displays a QR Code or Barcode from their e-wallet app, and the merchant uses a scanner or their own application to scan it and process the payment.[16]

The fact that Ksher designs its API based on these usage flows has significant implications for designing a NodeJS library. A good library should not have a single createPayment() method. Instead, it should have a structure that reflects this architecture, such as creating separate, clearly named methods like createRedirectOrder(), createNativeOrder(), and createQuickPayOrder(). This allows developers using the library to choose the appropriate integration model for their application correctly and without confusion. This guide will primarily focus on the **Redirect API**, as it is the most commonly used model for web applications developed with NodeJS.

Part 2: Essential Foundations: Authentication and Environment Setup

This section is the core of starting the integration process. It will clarify potentially confusing issues for developers, such as endpoint changes and different authentication methods, to build a solid and correct foundation before starting to code.

2.1. Requesting API Credentials

Before connecting to the Ksher API, merchants or developers must register for an account with Ksher.[9] The required steps and documents differ for individuals and legal entities.[9] The approval process typically takes 3-5 business days. After approval, the merchant will receive an

Account ID and Password to log into the Ksher Merchant Platform.[9]

Within this Merchant Platform, developers can access the crucial credentials used for API authentication, which include:

- **App ID:** The merchant's unique identifier used to identify them when calling the API.
- **Private Key:** A private key used to create a digital signature to verify that the request originates from the actual merchant and that the data has not been tampered with en route.

2.2. API Environment and Endpoints: Understanding Key Changes

One of the most significant challenges for developers starting with Ksher is the confusion regarding API URLs, or Endpoints. Many older documents and SDKs refer to the [specific_name].vip.ksher.net domain.[17] However, several recent documents have clearly stated that this domain

is being deprecated for new merchants and recommend using a new domain instead.[13]

This inconsistency is not a documentation error but a sign of a major technical migration that Ksher is undergoing. The existence of different domains (doc.vip.ksher.net, api.ksher.net, gateway.ksher.com) alongside different authentication methods (API Token for the old system and RSA Signature for the new one) is clear evidence that Ksher is moving from one API platform to a more secure and modern one.

Therefore, to ensure that the developed library will be usable and secure in the long term, **it is imperative to develop it based solely on the new API endpoints.** Connecting to the soon-to-be-deprecated .vip.ksher.net domain carries a high risk of system failure in the future.

To eliminate all confusion, the table below lists the correct and current endpoints for Ksher's core services that should be used for development.

Table: Correct and Current Ksher API Endpoints

Service	Environment	HTTP Method	URL
Gateway Pay (Create Payment)	Production	POST	https://gateway.ksher.com/api/gateway_pay
Order Query (Check Status)	Production	POST	https://gateway.ksher.com/api/gateway_order_query

Order Refund	Production	POST	https://gateway.ksher.com/api/gateway_refund
Sandbox (for testing)	Sandbox	POST	``

Note: The URL for refunds (gateway_refund) is inferred from the structure of other endpoints and should be confirmed with technical documentation or SDK analysis.

2.3. Signature-based Authentication Protocol

The current version of the Ksher API (using gateway.ksher.com) employs a highly secure authentication method called **Signature-based Authentication** [21], replacing the simpler

API_TOKEN method found in the older system.[20] This signature-based authentication is a complex protocol that requires precise calculation. Any minor error will cause the API to reject all requests immediately.

The process of creating a signature involves taking all parameters in the request payload, sorting them alphabetically, creating a string, and encrypting it with the merchant's Private Key using an asymmetric-key cryptography algorithm like RSA-SHA256. The result is the sign value, which is sent with the request for the Ksher server to verify its authenticity.

Ksher's transition to this protocol reflects its high priority on data security. The NodeJS library to be developed must therefore include a dedicated module for generating this signature accurately, which will be the most critical core of the library. The technical details and steps for signature generation will be explained in detail in Part 4.

Part 3: Core API Functions: An In-Depth Analysis

This section will detail the main endpoints that the library needs to implement, based on the most complete technical information gathered, to serve as a comprehensive reference for development.

3.1. The Payment Lifecycle

To provide an overview of how the different endpoints work together, a typical payment lifecycle through Ksher involves the following steps:

1. **Order Creation:** The merchant's website sends a request to the gateway_pay endpoint to create a payment order.
2. **User Redirection:** Ksher responds with a payment URL. The merchant then redirects the user's browser to that URL.
3. **Payment:** The user completes the payment on the Ksher or e-wallet provider's page.
4. **Webhook Notification:** Once the payment is successful, Ksher's system sends a server-to-server notification (Webhook) to a URL specified by the merchant.
5. **Status Query:** The merchant can send a request to the gateway_order_query endpoint to confirm the final status of the payment order (especially if a Webhook is not received).
6. **Settlement:** Ksher collects the funds and transfers them to the merchant's bank account within 1 business day.[6]

3.2. Endpoint Analysis: Creating a Payment (Redirect API)

This is the primary endpoint for online merchants who want to accept payments on their websites. The details are as follows:

- **Endpoint:** https://gateway.ksher.com/api/gateway_pay
- **HTTP Method:** POST

The request must include order and merchant information. Key parameters are merchant_order_id, which must be a unique reference ID in the merchant's system; amount, which must be an integer in the smallest currency unit (e.g., 150.50 THB must be sent as 15050); redirect_url for successful payments; and redirect_url_fail for failed payments.[13]

Upon a successful request, the API response will contain the crucial reference field, which is the URL of the payment page that the developer must use to redirect the user to the payment process.[13]

Table: Parameters for Creating a Payment Order (Create Order - Redirect API)

Parameter	Type	Required	Description

merchant_order_id	String	Yes	The merchant's order number, must be unique.
amount	Number	Yes	The amount in the smallest currency unit (e.g., 100 THB is 10000).
redirect_url	String	Yes	The URL to redirect to upon successful payment.
redirect_url_fail	String	Yes	The URL to redirect to upon failed payment.
channel	String	No	The specific payment channel, e.g., promptpay,truemoney.
note	String	No	Additional notes about the order.
timestamp	String	Yes	The time the request was created (Unix timestamp or other specified format).
appid	String	Yes	The App ID received from Ksher.
sign	String	Yes	The digital signature calculated from all

			parameters.

3.3. Endpoint Analysis: Querying Order Status

This endpoint is crucial for reconciliation and confirming the final status of a transaction.

- **Endpoint:** https://gateway.ksher.com/api/gateway_order_query
- **HTTP Method:** POST

The request must specify the mch_order_no (or ksher_order_no) along with authentication parameters. The response will indicate the status of that order, with error codes in case of issues, such as order not exist or sign failed to verify.[21]

3.4. Endpoint Analysis: Processing Refunds

The refund function is essential for customer service and merchant operations. Ksher provides a detailed endpoint for refunds.[22]

- **Endpoint:** https://gateway.ksher.com/api/gateway_refund (inferred)
- **HTTP Method:** POST

A refund request must specify the mch_order_no of the original transaction, a mch_refund_no which is a unique refund reference ID on the merchant's side, and the refund_fee, which is the amount to be refunded (partial refunds are supported). The response is clearly structured for success (result: "SUCCESS") and failure (result: "FAIL") cases, with codes and messages explaining any errors.[22]

Table: Key Parameters for Requesting a Refund (Refund Order API)

Parameter	Type	Required	Description
appid	String(32)	Yes	The merchant's App ID.
mch_order_no	String(32)	Yes	The original order

			number to be refunded.
mch_refund_no	String(32)	Yes	The merchant's refund number, must be unique.
refund_fee	Int	Yes	The amount to be refunded (in the smallest currency unit).
fee_type	String(16)	Yes	The currency, e.g., THB.
nonce_str	String(32)	Yes	A random string for added security.
time_stamp	String(256)	Yes	The time the request was created.
sign	String(256)	Yes	The digital signature.

3.5. Asynchronous Notifications: Using Webhooks

Relying solely on user redirection to confirm a payment is not reliable enough, as users might close their browser prematurely or experience network issues. Ksher provides a Webhook system, which sends a direct server-to-server notification from Ksher's server to the merchant's server when the payment status changes.[17]

Using Webhooks is essential for systems that require accuracy and stability. A good library should include guidance or helper functions for validating received Webhooks, especially for verifying the signature of the incoming data to prevent data tampering by malicious actors.

Part 4: Building the NodeJS Library: A Development Guide

This section will transform the theoretical knowledge from the previous parts into practical, usable code, presented as a step-by-step development guide.

4.1. Project Setup and Dependencies

Start by creating a new NodeJS project and installing the necessary libraries.

1. Create a directory for the project and navigate into it.
2. Run the command npm init -y to create a package.json file.
3. Install the necessary libraries:
    - axios: For sending HTTP requests to the Ksher API.
    - crypto: A built-in NodeJS module for cryptographic functions, which will be used to create the signature.

Bash

```
npm install axios
```

4.2. Library Structure

For organization and ease of maintenance, the library should be structured as follows:

```
ksher-nodejs-lib/
```

```
├── src/
│   ├── KsherClient.js   # Main class for managing configuration and API calls
│   ├── Signature.js     # Dedicated module for signature generation
│   ├── constants.js     # File for storing constants like endpoint URLs
│   └── errors.js        # Classes for custom error handling
├── index.js             # Main file for exporting the library
└── package.json
```

4.3. Developing the Authentication Layer

This is the most critical and complex part of the library: correctly generating the digital signature according to Ksher's protocol. The process in Signature.js will be as follows:

1. **Gather Parameters:** Receive the data object (payload) to be sent to the API.
2. **Filter Data:** Filter out parameters that have no value (null, undefined, or an empty string) and are not the sign parameter itself.
3. **Sort:** Sort the keys of the remaining parameters alphabetically (A-Z).
4. **Create String:** Concatenate the sorted parameters into a string in the format key1=value1&key2=value2&....
5. **Encrypt:** Use the crypto module of NodeJS to create a signature with the RSA-SHA256 algorithm, using the string from step 4 as the data and the merchant's Private Key for encryption.
6. **Format Result:** Convert the encryption result into a Hex String.

Example code in Signature.js:

JavaScript

```
const crypto = require('crypto');

function generateSignature(payload, privateKey) {
  // Steps 2-4: Filter, sort, and create the string
  const sortedKeys = Object.keys(payload).sort();
  const stringToSign = sortedKeys
    .filter(key => key!== 'sign' && payload[key]!== '' && payload[key]!== null && payload[key]!== undefined)
    .map(key => `${key}=${payload[key]}`)
```

```
    .join('&');

  // Steps 5-6: Create the signature and format the result
  const sign = crypto.createSign('RSA-SHA256');
  sign.update(stringToSign, 'utf-8');
  const signature = sign.sign(privateKey, 'hex');

  return signature;
}

module.exports = { generateSignature };
```

4.4. Developing API Wrapper Functions

In KsherClient.js, create methods to wrap each API endpoint call, making them easier to use.

JavaScript

```
const axios = require('axios');
const { generateSignature } = require('./Signature');
const { API_ENDPOINTS } = require('./constants');
const fs = require('fs');
const crypto = require('crypto');

class KsherClient {
  constructor(config) {
    this.appid = config.appid;
    this.privateKey = fs.readFileSync(config.privateKeyPath, 'utf8');
    this.baseURL = API_ENDPOINTS.PRODUCTION;
  }

  async _sendRequest(endpoint, data) {
    const payload = {
      ...data,
      appid: this.appid,
      nonce_str: crypto.randomBytes(16).toString('hex'), // Example of generating nonce_str
```

```
            time_stamp: Math.floor(Date.now() / 1000).toString()
        };

        payload.sign = generateSignature(payload, this.privateKey);

        try {
            const response = await axios.post(`${this.baseURL}${endpoint}`, payload);
            return response.data;
        } catch (error) {
            // Handle errors here
            throw error;
        }
    }

    createOrder(orderData) {
        return this._sendRequest('/api/gateway_pay', orderData);
    }

    queryOrder(mch_order_no) {
        const data = { mch_order_no };
        return this._sendRequest('/api/gateway_order_query', data);
    }

    refundOrder(refundData) {
        return this._sendRequest('/api/gateway_refund', refundData);
    }
}

module.exports = KsherClient;
```

4.5. Error Handling

A good library should have robust error handling, distinguishing between:

- **Network Errors:** Errors arising from connectivity issues, such as axios being unable to connect to the server.
- **API Errors:** Errors returned by the Ksher server, which can be identified from the code and result values in the JSON response body to determine the cause, such as an invalid signature, incomplete data, or a failed transaction.

Part 5: Practical Application and Advanced Topics

The final part will demonstrate how to use the library in a real-world web application and discuss security best practices.

5.1. Practical Usage Example (Express.js)

The following is an example of a web server built with Express.js, showing how to use the developed Ksher library.

JavaScript

```
const express = require('express');
const KsherClient = require('./ksher-nodejs-lib'); // Assuming the library is here

const app = express();
app.use(express.json());

const ksherClient = new KsherClient({
    appid: 'mch12345', // Your App ID
    privateKeyPath: './path/to/your_private_key.pem' // Path to your Private Key
});

// Route to create a payment order
app.post('/create-payment', async (req, res) => {
    try {
        const orderData = {
            amount: 10000, // 100.00 THB
            merchant_order_id: `ORDER_${Date.now()}`,
            redirect_url: 'http://localhost:3000/payment-callback',
            redirect_url_fail: 'http://localhost:3000/payment-failed'
        };
```

```
        const response = await ksherClient.createOrder(orderData);

        if (response.data && response.data.reference) {
            // Redirect the user to the payment page
            res.redirect(response.data.reference);
        } else {
            res.status(500).send('Failed to create payment order.');
        }
    } catch (error) {
        console.error(error);
        res.status(500).send('An error occurred.');
    }
});

// Route to handle the callback after payment
app.get('/payment-callback', async (req, res) => {
    const { mch_order_no } = req.query;
    try {
        const status = await ksherClient.queryOrder(mch_order_no);
        // Check the status and display the result to the user
        res.send(`Payment status for order ${mch_order_no}: ${status.data.result}`);
    } catch (error) {
        res.status(500).send('Failed to verify payment.');
    }
});

// Route to receive Webhooks
app.post('/webhook', (req, res) => {
    const webhookData = req.body;
    // **Important:** Webhook signature verification should be implemented here
    console.log('Webhook received:', webhookData);
    res.status(200).send('OK');
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

5.2. Security Best Practices

Ksher places a high priority on security, with systems compliant with PCI DSS standards and fraud prevention mechanisms.[6] Therefore, developers should adhere strictly to security principles:

- **Private Key Storage: Never** hard-code the Private Key directly in the source code. It should be stored securely, for example, using Environment Variables, AWS Secrets Manager, or HashiCorp Vault.
- **Webhook Verification:** Every time a Webhook is received, its accompanying signature must be verified to ensure the data genuinely comes from Ksher and has not been forged.
- **Use of nonce_str:** The nonce_str (number used once) parameter should be a newly generated random value for each request to prevent Replay Attacks.

Conclusion

This document has compiled all the necessary information for developing a NodeJS library to connect to the Ksher Payment Gateway correctly and completely. It has clarified key points that could cause confusion, such as changes in API endpoints and authentication protocols. It also presented a step-by-step guide to designing and developing the library, from signature generation to creating wrapper functions for the main API calls, and concluded with a practical usage example and security best practices. Following the guidelines in this document will enable developers to build a stable, secure library that can be confidently deployed in a production environment.

Appendix: API Reference

Table A: API Environment URLs

Environment	Base URL
Production	https://gateway.ksher.com
Sandbox	``

Table B: Endpoint for Creating an Order (Create Order)

- **Endpoint:** /api/gateway_pay
- **Method:** POST
- **Parameters:** See the table in section 3.2
- **Successful Response:** A JSON object containing data.reference as the payment URL.

Table C: Endpoint for Querying an Order (Query Order)

- **Endpoint:** /api/gateway_order_query
- **Method:** POST
- **Key Parameter:** mch_order_no
- **Response:** A JSON object with data.result indicating the status (e.g., SUCCESS, FAIL).

Table D: Endpoint for Refunding an Order (Refund Order)

- **Endpoint:** /api/gateway_refund
- **Method:** POST
- **Parameters:** See the table in section 3.4
- **Response:** A JSON object with data.result indicating the refund status.

Table E: Common Error Codes (from Order Query)

Code (code)	Meaning
-1	merchant info error
-2	sign failed to verify
-3	order not exist
-4	order number is not unique
-100	query order failed