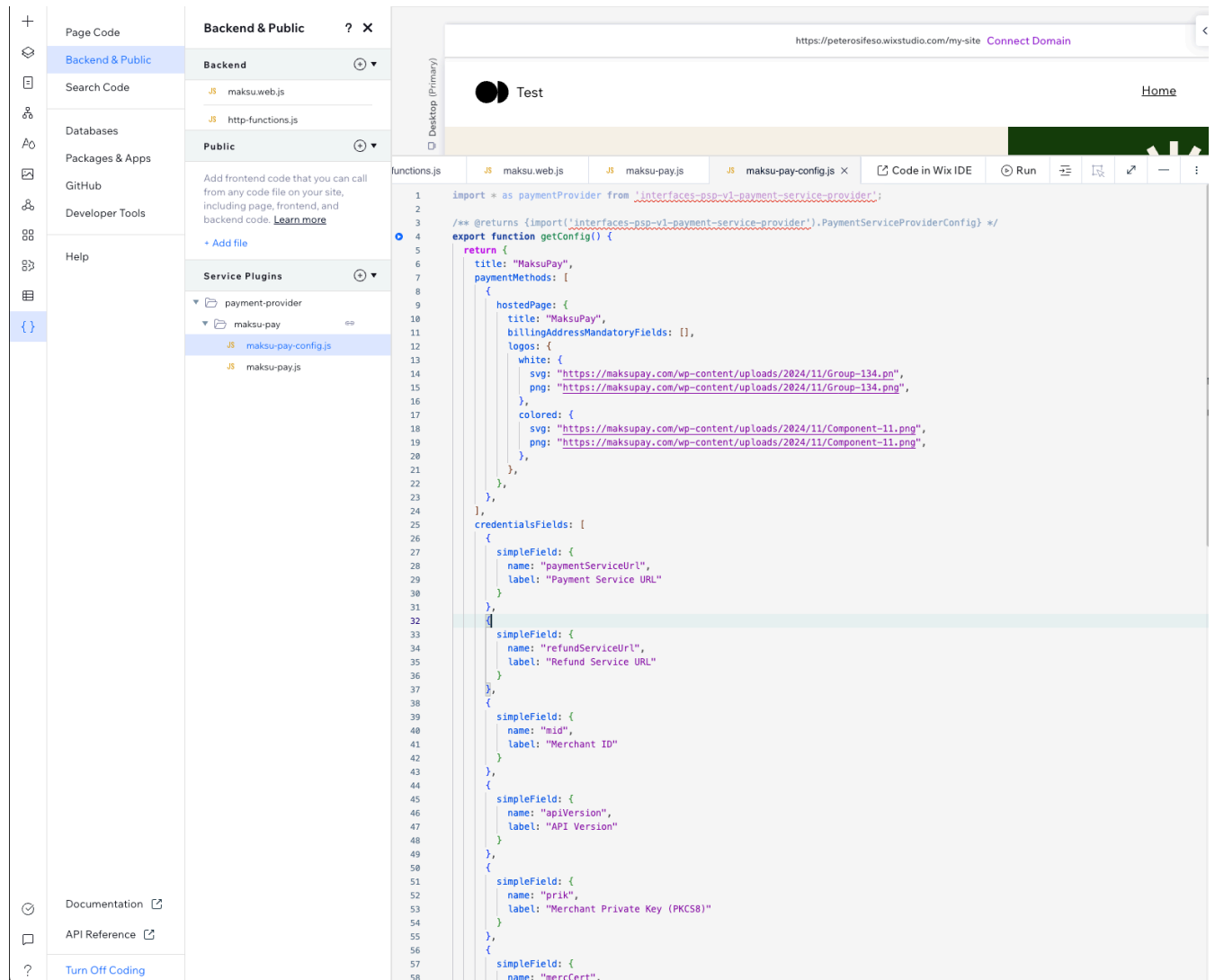


MaksuPay Wix Custom Payment Plugin

A dark rectangular button with a white pencil icon and the text "Edit Site".

From the wix dashboard, click on “Edit Site” button on the left bottom corner to start editing your wix website.

The screenshot shows the Wix Studio IDE interface. On the left is a sidebar with navigation options: Page Code, Backend & Public (selected), Search Code, Databases, Packages & Apps, GitHub, Developer Tools, and Help. The 'Backend & Public' section is expanded, showing a 'Backend' folder with 'maksu.web.js' and 'http-functions.js', and a 'Public' folder. Under 'Service Plugins', the 'maksu-pay' plugin is selected, showing 'maksu-pay-config.js' and 'maksu-pay.js'. The main editor area displays the 'maksu-pay-config.js' file, which contains a TypeScript configuration for the MaksuPay payment plugin. The code includes imports for the payment service provider and a 'getConfig()' function that returns an object with plugin details like title, logos, hosted page information, and credential fields for payment, refund, and merchant identification. The browser preview at the top shows a 'Test' button and a 'Home' link.

```
1 import * as paymentProvider from 'interfaces-psp-v1-payment-service-provider';
2
3 /** @returns {import('interfaces-psp-v1-payment-service-provider').PaymentServiceProviderConfig} */
4 export function getConfig() {
5   return {
6     title: "MaksuPay",
7     paymentMethods: [
8       {
9         hostedPage: {
10           title: "MaksuPay",
11           billingAddressMandatoryFields: [],
12           logos: {
13             white: {
14               svg: "https://maksupay.com/wp-content/uploads/2024/11/Group-134.png",
15               png: "https://maksupay.com/wp-content/uploads/2024/11/Group-134.png",
16             },
17             colored: {
18               svg: "https://maksupay.com/wp-content/uploads/2024/11/Component-11.png",
19               png: "https://maksupay.com/wp-content/uploads/2024/11/Component-11.png",
20             },
21           },
22         },
23       },
24     ],
25     credentialsFields: [
26       {
27         simpleField: {
28           name: "paymentServiceUrl",
29           label: "Payment Service URL"
30         },
31       },
32       {
33         simpleField: {
34           name: "refundServiceUrl",
35           label: "Refund Service URL"
36         },
37       },
38       {
39         simpleField: {
40           name: "mid",
41           label: "Merchant ID"
42         },
43       },
44       {
45         simpleField: {
46           name: "apiVersion",
47           label: "API Version"
48         },
49       },
50     ],
51     {
52       simpleField: {
53         name: "prik",
54         label: "Merchant Private Key (PKCS8)"
55       },
56     },
57     {
58       simpleField: {
59         name: "mercCert",
```

Create a service plugin



In the wix studio, select the code icon menu, click Backend & Public

Backend & Public

. In the Service Plugins Click on the + icon button to create a new plugin, select “Payment” and enter the name “maksu-pay” or customize as needed and finally click the “Add & Edit Code” button.

The above steps adds a folder with the name “maksu-pay” and two new files within the folder

1. maksu-pay-config.js
2. maksu-pay.js

1. maksu-pay-config.js

This file contains a function named “getConfig” which helps to define the credentials needed for processing payments from the merchant. This includes:

1. The payment service url
2. The refund service url
3. The merchant id
4. The api version. Currently supporting “5”. Note that “5.0” causes an error in the refund json API.
5. The merchant private key
6. The merchant certificate
7. The processor certificate

Update logo images

The logos in this configuration should be updated with links to the right images to properly display the MaksuPay logo in the payment settings as well as the checkout page accessible by customers.

The file should be updated like below:

```
import * as paymentProvider from 'interfaces-psp-v1-payment-service-provider';

/** @returns
{import('interfaces-psp-v1-payment-service-provider').PaymentServiceProviderConfig} */
export function getConfig() {
  return {
    title: "MaksuPay",
    paymentMethods: [
      {
        hostedPage: {
```

```
    title: "MaksuPay",
    billingAddressMandatoryFields: [],
    logos: {
      white: {
        svg: "https://maksupay.com/wp-content/uploads/2024/11/Group-134.pn",
        png: "https://maksupay.com/wp-content/uploads/2024/11/Group-134.png",
      },
      colored: {
        svg: "https://maksupay.com/wp-content/uploads/2024/11/Component-11.png",
        png: "https://maksupay.com/wp-content/uploads/2024/11/Component-11.png",
      },
    },
  },
],
credentialsFields: [
  {
    simpleField: {
      name: "paymentServiceUrl",
      label: "Payment Service URL"
    }
  },
  {
    simpleField: {
      name: "refundServiceUrl",
      label: "Refund Service URL"
    }
  },
  {
    simpleField: {
      name: "mid",
      label: "Merchant ID"
    }
  },
  {
    simpleField: {
      name: "apiVersion",
      label: "API Version"
    }
  },
  {
    simpleField: {
```

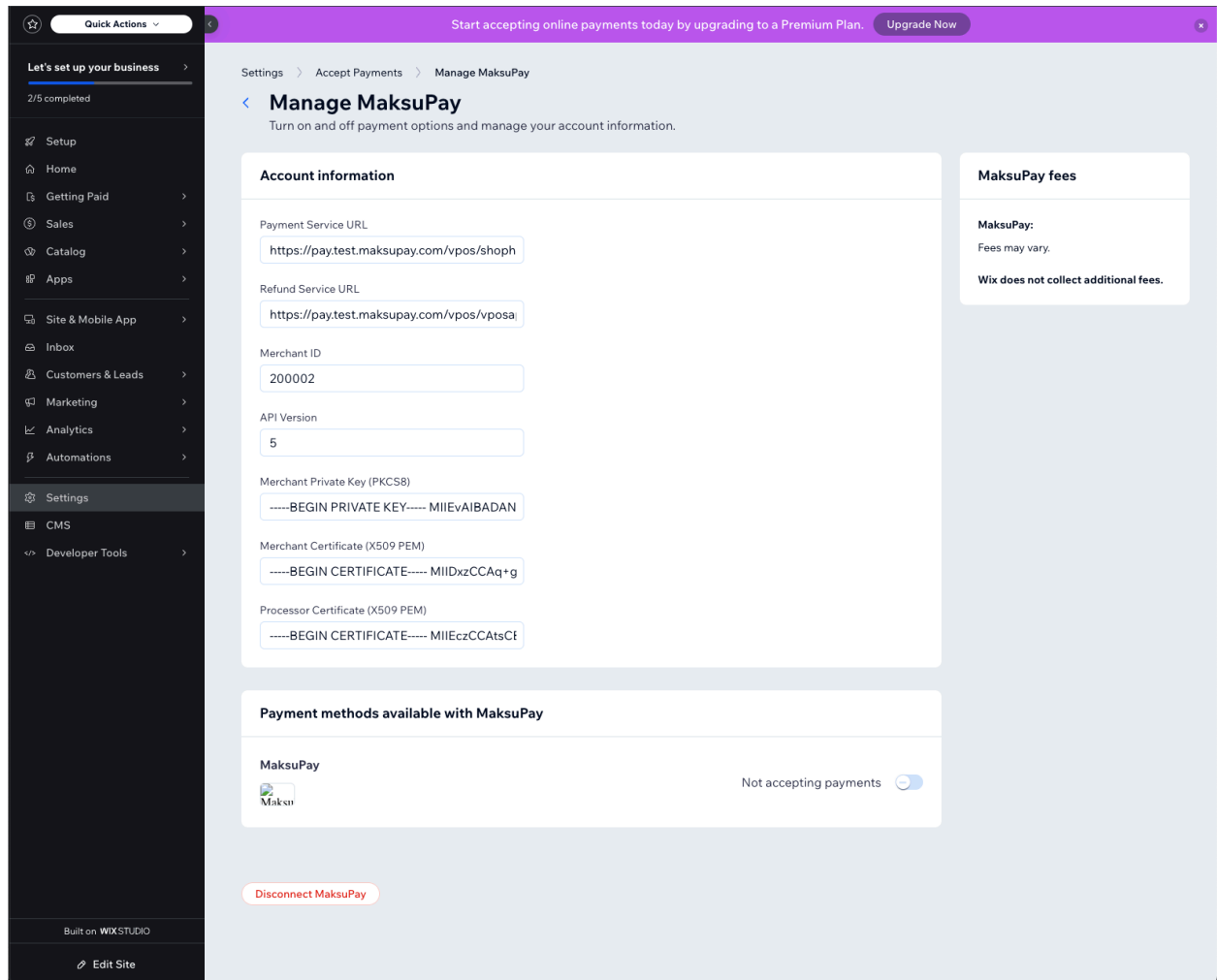
```

        name: "prik",
        label: "Merchant Private Key (PKCS8) "
    },
    {
        simpleField: {
            name: "mercCert",
            label: "Merchant Certificate (X509 PEM) "
        }
    },
    {
        simpleField: {
            name: "procCert",
            label: "Processor Certificate (X509 PEM) "
        }
    }
],
};
}

```

This configuration generates a page on the wix dashboard that can be accessed in:
Dashboard -> Settings -> Accept Payments -> MaksuPay -> Connect

This will allow the merchant to enter their Maksu credentials to enable this payment method.
NB: The `connectAccount` function in maksu-pay.js must be properly set up to be able to successfully connect the payment method from the wix dashboard.



A comprehensive guide to the details on this page is available on the wix developer docs: [“https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/get-config”](https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/get-config).

2. maksu-pay.js

This file contains 3 functions:

1. connectAccount
2. createTransaction
3. refundTransaction

1. ConnectAccount

This function is run when the user tries to connect the “MaksuPay” payment method from the dashboard. Essentially, it saves the credentials for future access when payments and refunds are to be made.

The connectAccount function for MaksuPay should be updated as below:

```
export const connectAccount = async (options, context) => {
  const {credentials} = options;
  return {
    credentials
  };
};
```

More information about this function is available on the wix developer documentation:

[“https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/connect-account”](https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/connect-account)

2. createTransaction

This function is called when a user checks out their order from the wix shop platform. It has access to the merchant credentials and expects the payment API to return a redirectUrl and a transaction id when the transaction is created successfully on the payment platform. The redirect url is essentially where the user trying to make the payment for their order on wix is redirected to.

The createTransaction function for MaksuPay should be updated as below:

```
export const createTransaction = async (options, context) => {
  const {merchantCredentials, order} = options;

  try {
    const response = await startPayment({
      merchantId: merchantCredentials.mid,
      apiVersion: merchantCredentials.apiVersion,
      mercCert: merchantCredentials.mercCert,
      procCert: merchantCredentials.procCert,
      privateKey: merchantCredentials.prik,
      serviceUrl: merchantCredentials.paymentServiceUrl,
      order: order
    });
    return {
      pluginTransactionId: response.transactionId,

```

```

        redirectUrl: response.redirectUrl
    }
} catch (error) {
    return {
        errorCode: '', // MaksuPay does not support this at the moment
        errorMessage: error?.message || 'Something went wrong'
    }
}
};

```

The actual logic is defined on the wix backend in a file named “[maksu.web.js](#)”.

More information about this function is available in the wix developer documentation:

“<https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/create-transaction>”

3. refundTransaction

This function is called when the owner or manager of the wix shop platform decides to refund a users’ order. This function also has access to the merchant credentials and expects the API to return a refund id when the transaction is successful.

The refundTransaction function for MaksuPay should be updated as below:

```

export const refundTransaction = async (options, context) => {
    const {merchantCredentials, pluginTransactionId, wixRefundId, refundAmount} =
options;
    const {currency} = context;
    try {
        const response = await refundPayment({
            refundId: wixRefundId,
            merchantId: merchantCredentials.mid,
            apiVersion: merchantCredentials.apiVersion,
            mercCert: merchantCredentials.mercCert,
            procCert: merchantCredentials.procCert,
            privateKey: merchantCredentials.prik,
            orderId: pluginTransactionId,
            refundAmount: refundAmount,
            currency: currency,
            serviceUrl: merchantCredentials.refundServiceUrl
        });
        return {pluginRefundId: response.refundId };
    } catch(error) {
        return {

```

```

        errorCode: '', // MaksuPay does not support this at the moment
        errorMessage: error?.message || 'Something went wrong'
    }
}
};

```

Similar to the createTransaction function, the actual logic is defined on the wix backend in a file named [“maksu.web.js”](#).

More information about this function is available in the wix developer documentation: [“https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/refund-transaction”](https://dev.wix.com/docs/velo/events-service-plugins/payments/service-plugins/wix-payments/payment-provider/refund-transaction)

Overview of maksu-pay.js

The maksu-pay.js file should be similar to:

```

import * as paymentProvider from 'interfaces-psp-v1-payment-service-provider';
import { startPayment, refundPayment } from 'backend/maksu.web';

/**
 * This payment plugin endpoint is triggered when a merchant provides required data to
 * connect their PSP account to a Wix site.
 * The plugin has to verify merchant's credentials, and ensure the merchant has an
 * operational PSP account.
 * @param {import('interfaces-psp-v1-payment-service-provider').ConnectAccountOptions}
options
 * @param {import('interfaces-psp-v1-payment-service-provider').Context} context
 * @returns
{Promise<import('interfaces-psp-v1-payment-service-provider').ConnectAccountResponse |
import('interfaces-psp-v1-payment-service-provider').BusinessError>}
 */
export const connectAccount = async (options, context) => {
    const {credentials} = options;
    return {
        credentials
    };
};

/**
 * This payment plugin endpoint is triggered when a buyer pays on a Wix site.

```



```

* The plugin has to process this payment request but prevent double payments for the
same `wixTransactionId`.
* @param
{import('interfaces-psp-v1-payment-service-provider').CreateTransactionOptions}
options
* @param {import('interfaces-psp-v1-payment-service-provider').Context} context
* @returns
{Promise<import('interfaces-psp-v1-payment-service-provider').CreateTransactionRespon
se | import('interfaces-psp-v1-payment-service-provider').BusinessError>}
*/
export const createTransaction = async (options, context) => {
  const {merchantCredentials, order} = options;

  try {
    const response = await startPayment({
      merchantId: merchantCredentials.mid,
      apiVersion: merchantCredentials.apiVersion,
      mercCert: merchantCredentials.mercCert,
      procCert: merchantCredentials.procCert,
      privateKey: merchantCredentials.prik,
      serviceUrl: merchantCredentials.paymentServiceUrl,
      order: order
    });
    return {
      pluginTransactionId: response.transactionId,
      redirectUrl: response.redirectUrl
    }
  } catch (error) {
    return {
      errorCode: '', // MaksuPay does not support this at the moment
      errorMessage: error?.message || 'Something went wrong'
    }
  }
};
/**
* This payment plugin endpoint is triggered when a merchant refunds a payment made on
a Wix site.
* The plugin has to process this refund request but prevent double refunds for the
same `wixRefundId`.
* @param
{import('interfaces-psp-v1-payment-service-provider').RefundTransactionOptions}
options

```

```

* @param {import('interfaces-psp-v1-payment-service-provider').Context} context
* @returns
{Promise<import('interfaces-psp-v1-payment-service-provider').CreateRefundResponse |
import('interfaces-psp-v1-payment-service-provider').BusinessError>}
*/

export const refundTransaction = async (options, context) => {
  const {merchantCredentials, pluginTransactionId, wixRefundId, refundAmount} =
options;
  const {currency} = context;
  try {
    const response = await refundPayment({
      refundId: wixRefundId,
      merchantId: merchantCredentials.mid,
      apiVersion: merchantCredentials.apiVersion,
      mercCert: merchantCredentials.mercCert,
      procCert: merchantCredentials.procCert,
      privateKey: merchantCredentials.prik,
      orderId: pluginTransactionId,
      refundAmount: refundAmount,
      currency: currency,
      serviceUrl: merchantCredentials.refundServiceUrl
    });
    return {pluginRefundId: response.refundId };
  } catch(error) {
    return {
      errorCode: '', // MaksuPay does not support this at the moment
      errorMessage: error?.message || 'Something went wrong'
    }
  }
};

```

Backend (maksu.web.js)

This file contains the actual logic for creating and refunding transactions as well as verification of the responses from maksupay servers.

Create a backend file

To create a backend file in the wix studio, select the code icon menu, Backend & Public. On the Backend section click the + icon and “Add web module”, name the new file “maksu.web.js”

Overview of the maksu.web.js file

***Make sure to replace YOUR_DOMAIN with the appropriate link**

```
import { Permissions, webMethod } from "wix-web-module";
import { fetch } from 'wix-fetch'
import forge from 'node-forge';
import * as jsdom from 'jsdom';

const YOUR_DOMAIN = 'https://site-name.wixstudio.com/my-site';

export const startPayment = webMethod(
  Permissions.Anyone,
  async ({merchantId, apiVersion, mercCert, procCert, privateKey, serviceUrl, order})
=> {
  const formData = createFormDataFromOrder(order, merchantId, apiVersion);
  const signatureDataString = createPaymentSignatureDataString(formData);
  formData.publicKeyHash = getPublicKeyHashFromCert(mercCert);
  formData.signature = createDigitalSignature(privateKey, signatureDataString);
  const response = await fetch(serviceUrl, {
    headers:{
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    method: 'POST',
    body: new URLSearchParams(formData).toString()
  });
  const transactionId = response.url?.split('trid=')[1]?.split('&')[0];
  if (transactionId) {
    return {transactionId: transactionId, redirectUrl: response.url};
  }
  const htmlString = await response.text();
  const errorMessage = new
jsdom.JSDOM(htmlString).window.document.getElementById('errors')?.textContent ||
'Something went wrong';
  throw Error(errorMessage);
}
);

export const refundPayment = webMethod(
  Permissions.Admin,
```

```

    async({refundId, merchantId, apiVersion, certificate, procCert, privateKey, orderId,
refundAmount, currency, serviceUrl}) => {
        const timeStamp = formatDateToTimestamp(new Date());
        const body = {"message" : {"refundReq" : {"merchantId" : merchantId, "orderInfo" :
{"orderAmount" : refundAmount, "currency" : currency, "orderId" : orderId}}, "id" :
`M${refundId.split('-').join('')}` , "version" : apiVersion, "ts" : timeStamp,
"senderId" : merchantId}};
        const publicKeyHash = getPublicKeyHashFromCert(certificate);
        const signature = createDigitalSignature(privateKey, JSON.stringify(body));
        const response = await fetch(serviceUrl, {
            headers:{
                'Content-Type': 'application/json',
                'X-Sender-ID': merchantId,
                'X-Public-Key-Hash': publicKeyHash,
                'X-Payload-Signature': 'RSA-SHA256;' + signature
            },
            method: 'POST',
            body: JSON.stringify(body)
        });

        try {
            return response.buffer().then((buffer) => {
                const responseSignatureString =
response.headers.get('X-Payload-Signature')?.split('RSA-SHA256;')[1];
                const responsePublicKeyHash = response.headers.get('X-Public-Key-Hash');
                const isValidResponse = verifyRefundResponse(responseSignatureString,
responsePublicKeyHash, procCert, buffer);
                if (isValidResponse) {
                    // convert buffer to json and return refund id
                    const jsonString = new TextDecoder('utf-8').decode(buffer);
                    let json = JSON.parse(jsonString);
                    const apiRes = json.message?.refundRes;
                    if (apiRes?.status === 'CAPTURED') {
                        return {refundId: apiRes?.txId};
                    } else {
                        throw Error(apiRes?.description);
                    }
                } else {
                    throw Error('The refund response from the server could not be verified')
                }
            });
        } catch(error) {

```

```

        console.error('Error processing refund:', error?.message);
        throw error;
    }
});

const createDigitalSignature = (privateKey, data) => {
    const privateKeyObj = forge.pki.privateKeyFromPem(privateKey);
    const md = forge.md.sha256.create();
    md.update(data, 'utf8');
    const signature = privateKeyObj.sign(md);
    return forge.util.encode64(signature);
};

const getPublicKeyHashFromCert = (certificate) => {
    try {
        const cert = forge.pki.certificateFromPem(certificate);
        const publicKey = cert.publicKey;
        const publicKeyAsn1 = forge.pki.publicKeyToAsn1(publicKey);
        const publicKeyDer = forge.asn1.toDer(publicKeyAsn1).getBytes();

        const md = forge.md.sha256.create();
        md.update(publicKeyDer);

        return forge.util.encode64(md.digest().getBytes());
    } catch (error) {
        console.error('Error processing certificate:', error.message);
        throw error;
    }
};

const verifyPaymentResponse = (responseSignatureBase64, responsePublicKeyHashBase64,
procCert, responseJson) => {
    let dataString = '';
    Object.keys(responseJson).filter(key => (key !== 'signature' && key !==
'publicKeyHash')).forEach(key => {
        if (responseJson[key]) {
            dataString += `${responseJson[key]}`;
        }
    });
    const md = forge.md.sha256.create();
    md.update(dataString, 'utf8');

```

```

    return verifyResponseMessageDigest(responseSignatureBase64,
responsePublicKeyHashBase64, procCert, md);
};

const verifyRefundResponse = (responseSignatureBase64, responsePublicKeyHashBase64,
procCert, responseArrayBuffer) => {
    const md = forge.md.sha256.create();
    const arrayBufferBinaryString = String.fromCharCode(...new
Uint8Array(responseArrayBuffer))
    md.update(arrayBufferBinaryString, 'binary');
    return verifyResponseMessageDigest(responseSignatureBase64,
responsePublicKeyHashBase64, procCert, md);
};

const verifyResponseMessageDigest = (responseSignatureBase64,
responsePublicKeyHashBase64, procCert, messageDigest) => {
    const procCertPublicKeyHashBase64 = getPublicKeyHashFromCert(procCert);

    if (procCertPublicKeyHashBase64.toLowerCase() !==
responsePublicKeyHashBase64.toLowerCase()) {
        console.error('Public key hash mismatch! Data may be spoofed.');
```

return false;

```
    }

    const signatureBytes = forge.util.decode64(responseSignatureBase64);

    const certificate = forge.pki.certificateFromPem(procCert);
    const publicKey = certificate.publicKey;
    const isValidSignature = publicKey.verify(messageDigest.digest().bytes(),
signatureBytes);

    if (!isValidSignature) {
        console.error('Invalid signature. Data may have been tampered with.');
```

return false;

```
    }
    return true;
};

const createPaymentSignatureDataString = (formData) => {
    let dataString = '';
    Object.keys(formData).forEach(key => {
        if (formData[key]) {
```

```

        dataString += `${formData[key]}`;
    }
});
return dataString;
};

const formatDateToTimestamp = (date) => {
    return (
        [
            date.getFullYear(),
            padTwoDigits(date.getUTCMonth() + 1),
            padTwoDigits(date.getUTCDate()),
        ].join('-') +
        " " +
        [
            padTwoDigits(date.getUTCHours()),
            padTwoDigits(date.getUTCMinutes()),
            padTwoDigits(date.getUTCSeconds()),
        ].join(":") + '+0000'
    );
    function padTwoDigits(num) {
        return num.toString().padStart(2, '0');
    }
}

const formatOrderItemsForMaksu = (orderItems) => {
    return 'items:' + JSON.stringify(orderItems.map(orderItem => ({
        't': 'p',
        'n': orderItem.name,
        'c': orderItem._id,
        'q': orderItem.quantity,
        'qu': '',
        'up': orderItem.price,
        'tt': 0,
        'tr': 0,
        'tp': orderItem.price * orderItem.quantity
    })))));
}

const createFormDataFromOrder = (order, merchantId, apiVersion) => {
    return (
        {

```

```

    version: `${apiVersion}`,
    mid: `${merchantId}`,
    lang: `${order.description.buyerInfo.buyerLanguage}`,
    orderId: `WX${order._id.split('-').join('').toUpperCase()}`,
    orderDesc: formatOrderItemsForMaksu(order.description.items),
    orderAmount: order.description.totalAmount / 100,
    currency: order.description.currency,
    payerName: `${order.description.shippingAddress.firstName}
${order.description.shippingAddress.lastName}`,
    payerEmail: order.description.shippingAddress.email,
    payerPhone: order.description.shippingAddress.phone,
    billCountry: order.description.billingAddress.countryCode,
    billState: order.description.billingAddress.city,
    billZip: order.description.billingAddress.zipCode,
    billCity: order.description.billingAddress.city,
    billAddress: order.description.billingAddress.address,
    shipState: order.description.shippingAddress.city,
    shipZip: order.description.shippingAddress.zipCode,
    shipCity: order.description.shippingAddress.city,
    shipAddress: order.description.shippingAddress.address,
    // weight: null,
    // dimensions: null,
    confirmUrl: `${YOUR_DOMAIN}/_functions/maksuPayResponseSuccess`,
    cancelUrl: `${YOUR_DOMAIN}/_functions/maksuPayResponseCancel`
  });
}

```

Required npm libraries

The backend depends on the following node libraries that must be installed on wix studio:

1. node-forge: Used for cryptographic operations. v1.3.1 was used.
2. jsdom: Used to parse and extract the error message from the html response when there is an error response when creating a transaction. v26.0.0 was used.

Installing node libraries

To install the node libraries, go to code, packages & apps and click the + icon in the npm section then “install npm package”, search for the package, click install and wait for the installation process to complete.

Web Methods

Wix web methods are accessible from the plugin (maksu-pay.js) file while being executed on the backend side without exposing sensitive data to the client.

1. **startPayment:** This method uses order information from the wix platform and the merchant credentials from the payment plugin to send a request to maksupays' service API in the expected format including a signature for the sent data and the merchants public key hash for data verification on the maksupay server and returns the transaction id and the return url to the payment plugin on success. In case of an error, the method throws an error message to the payment plugin and an error popup message is shown to the user by wix. Note that the signature and public key hash in this case are sent in the request params.
2. **refundPayment:** This method uses order information from the wix platform and the merchant credentials from the payment plugin to send a request to the maksupay refund JSON API in the expected format including a signature for the sent data and the merchants public key hash for data verification on maksupay server and returns the transaction id and the return url to the payment plugin on success. In case of an error, the method throws an error message to the payment plugin and an error popup message is shown to the user by wix. Note that the signature and public key hash in this case are sent in the request headers.

Other Helper Functions

1. **createDigitalSignature:** This function creates a digital signature by signing a given string with the merchants private key.
2. **getPublicKeyhashFromCert:** This function generates the merchant's public key hash from the merchant certificate.
3. **verifyPaymentResponse:** This function verifies the request params in the redirect url posted from the maksupay server on success or cancel after processing payment.
4. **verifyRefundResponse:** This function verifies the json response from a refund request is untampered with by comparing the processor public key hashes with the response public key hash and using the processors public key to verify the response message. In the case of refunds, the response array buffer response is used for verification as there seems to be issues with verifying the json response directly.
5. **verifyResponseMessageDigest:** This verification function is used to avoid code repetition in 3 and 4 above.
6. **createPaymentSignatureDataString:** This function creates the string which is signed with the merchants private key when creating payments.
7. **formatDateToTimestamp:** This function creates a UTC timestamp in the format "2025-04-11 10:36 :32+0000". This is used in the refund request body.
8. **formatOrderItemsForMaksu:** This function formats wixs' order items into maksus' items format.

9. `createFormdataFromOrder`: This function creates the request params for creating a payment transaction through maksupays' payment API.

HTTP Functions

Wix http function is required to process the return url request from maksupay after a payment is confirmed or canceled. This is because by default wix urls process only GET requests.

However, maksupay sends a POST request with request parameters including a signature and public key hash that can be used for verifying the response. Wix http functions allow us to react and process this POST request.

Creating an http-functions file

To create a backend file in the wix studio, select the code icon menu, Backend & Public. On the Backend section click the + icon and "Expose site API", an `http-functions.js` file will be created

More information on wix http functions:

<https://dev.wix.com/docs/velo/velo-only-apis/wix-http-functions/post>

Overview of the `http-functions.js` file

***Make sure to replace `YOUR_DOMAIN` with the appropriate link**

***Make sure to create the `/payment-success` and `/payment-cancel` pages on wix**

```
import { response } from "wix-http-functions";

const YOUR_DOMAIN = 'https://site-name.wixstudio.com/my-site'

export function post_maksuPayResponseSuccess(request) {
  const url = `${YOUR_DOMAIN}/payment-success`;
  return request.body.text().then((requestBody) => {
    const jsonRes = urlParamsToJson(requestBody);
    // TODO: Probably should verify that the response is ok but I do not have
    access to the plugin procert at this level AFAIK
    return redirectToUrl(url);
  });
}

export function post_maksuPayResponseCancel(request) {
  const url = `${YOUR_DOMAIN}/payment-cancel`;
  return redirectToUrl(url);
}

function redirectToUrl(url) {
```

```
    return response({
      status: 301,
      headers: {
        location: url
      }
    });
  }

function urlParamsToJson(urlParamsString) {
  const params = new URLSearchParams(urlParamsString);
  const json = {};
  for (const [key, value] of params.entries()) {
    json[key] = value;
  }
  return json;
}
```

Mak

The urlParamsToJson helper function helps to convert the request param in the body of the request to JSON for easy use.