Project Report: Implementation of ZipPay's Payment Gateway using Node.js

(By Gerard Mendjemo)

Introduction

This report provides a comprehensive overview of the implementation of ZipPay's payment gateway using Node.js. The project is structured into a backend API that handles payment requests and a frontend application developed using React to interact with the API. The backend includes secure data encryption and decryption mechanisms for transaction data, and the frontend facilitates user input and displays transaction outcomes.

Note: For instructions on how to install and run the project, please refer to the repo's README

Project Structure

The project is organized into multiple directories to separate concerns and maintain a clean codebase. Here is an overview of the project structure:

```
ZIPPAY_INTEGRATION/
|-- backend/
|   |-- KEYS/
|   |-- node_modules/
|   |-- routes/
|   |-- .gitignore
|   |-- package-lock.json
|   |-- package.json
|   |-- server.js
|   |-- utils.js
|-- frontend/
|   |-- node_modules/
|   |-- public/
|   |-- src/
|   |-- .gitignore
|   |-- package-lock.json
|   |-- package.json
|-- zippayAPI/
|   |-- KEYS/
|   |-- node_modules/
|   |-- resources/
|   |-- .gitignore
|   |-- package-lock.json
|   |-- package.json
```

```
|   |-- server.js
|   |-- utils.js
|-- docs/
|-- .env
|-- zippay对接文档.docx
|-- zippay对接文档.zh-CN.en.docx
```

Backend Implementation

`server.js`

The `server.js` file is the main entry point of the backend server. It sets up the Express server, applies middleware for parsing JSON and handling CORS, and defines the routes for handling payment requests and callbacks.

Key functionalities:

- Handles payment requests (Payin, Payout).
- Manages callback processing.
- Ensures secure communication through data encryption and decryption.

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const paymentPayout = require('./routes/exchangeOrder');
const paymentPayin = require('./routes/rechargeOrder');
const merchantQuery = require('./routes/merchantQuery');
const exchangeOrderQuery = require('./routes/exchangeOrderQuery');
const rechargeOrderQuery = require('./routes/rechargeOrderQuery');

const app = express();
const PORT = 5000;

app.use(bodyParser.json());
app.use(cors()); // For tests only

// Payment Request Endpoint
app.use('/payment/payin', paymentPayin);

// Collection Inquiry Endpoint
app.use('/payment/rechargeOrderquery', rechargeOrderQuery);

// Balance Inquiry Endpoint
```

```
app.use('/payment/merchantQuery', merchantQuery);

// Payment Endpoint
app.use('/payment/payout', paymentPayout);

// Payment Inquiry Endpoint
app.use('/payment/exchangeOrderquery', exchangeOrderQuery);

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
•
```

utils.js

The `utils.js` file contains utility functions for encryption, decryption, and constructing the data signature. These utilities are crucial for maintaining secure communication with ZipPay's API.

```
const NodeRSA = require('node-rsa');
const fs = require('fs');

const privateKey = fs.readFileSync('./KEYS/PRIVATE.pem', 'utf8');
const publicKey = fs.readFileSync('./KEYS/PUBLIC.pem', 'utf8');

const privateKeyObject = new NodeRSA(privateKey, 'pkcs8-private-pem');
const publicKeyObject = new NodeRSA(publicKey, 'pkcs8-public-pem');

const encryptData = (data) => {
    return privateKeyObject.encryptPrivate(Buffer.from(JSON.stringify(data)),
'base64');
};

const decryptData = (data) => {
    return publicKeyObject.decryptPublic(Buffer.from(data, 'base64'), 'utf8');
};

const constructSignData = (data) => {
    const sortedKeys = Object.keys(data).sort();
    return sortedKeys.map(key => data[key]).join('');
};

const cleanUpDecryptedData = (data) => {
    let decryptedSign = "";
    for (let character of data) {
```

```
        if (character !== " " && character !== '"') {
            decryptedSign += character;
        }
    }

    return decryptedSign;
}

module.exports = {
    encryptData,
    decryptData,
    constructSignData,
    cleanUpDecryptedData
};
```

Mock Server

The mock server (`zippayAPI/server.js`) simulates the behavior of ZipPay's API for testing purposes. It processes payout requests, verifies signatures, and sends callbacks to the backend.

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const PORT = 8080;
const axios = require('axios');
const { encryptData, decryptData, constructSignData, cleanUpDecryptedData } =
require('./utils');

app.use(bodyParser.json());
app.use(cors()); // For tests only

app.post('/payment/payout', (req, res) => {
    const payLoad = req.body;

    // Start decryption
    const signData = payLoad.merchantId + payLoad.merchantOrderId +
payLoad.amount + payLoad.nonce + payLoad.timestamp;

    const isValidSign = cleanUpDecryptedData(decryptData(payLoad.sign)) ==
signData;

    if (isValidSign) {
```

```
        let isPaymentSuccessful = false;
        let utr = '';

        // process payment and bank returns utr if possible

        // Scenario on successful transaction
        { isPaymentSuccessful = true; }

        // Update database (e.g resources -> exchange-order-list.txt

        if (isPaymentSuccessful) {
            // Simulate callback to the backend
            let data = {
                code: 1,
                msg: 'SUCCESS',
                merchantOrderId: payLoad.merchantOrderId,
                platformOrderId: "A22308231907420000184", // Default for now
                utr: utr
            }
            // Remove empty fields
            Object.keys(data).forEach(key => {
                if (data[key] === undefined || data[key] === null || data[key]
=== '') {
                    delete data[key];
                }
            });

            const sign = encryptData(constructSignData(data));

            data["sign"] = sign;

            // Check if callback was successfull
            axios.post('http://localhost:5000/payment/payout/callback', data)
                .then((response) => {
                    console.log('Payout callback sent successfully');
                }).catch((err) => {
                    console.error('Error sending callback:', err.message);
            });

            // Send response with confirmed payment
            res.status(200).json({
                code: 200,
                msg: 'SUCCESS',
                success: true,
                data: {
```

```
                    merchantId: payLoad.merchantId,
                    merchanOrderId: payLoad.merchanOrderId,
                    platformOrderId: payLoad.platformOrderId,
                    timestamp: `${Date.now()}`
                }
            });
            console.log("Payout was successful");

        } else {
            console.log("Payout was unsuccessful");
        }

    } else {
        res.status(400).send('invalid signature');
    }
});

app.listen(PORT, () => {
    console.log(`Mock server is running on port ${PORT}`);
});
```

Frontend Implementation

`App.js`

The `App.js` file sets up the React application with routing. It includes routes for the payout form and the success page.

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import PayoutForm from './components/PayoutForm';
import SuccessPage from './pages/SuccessPage';

const App = () => {
    return (
        <Router>
            <Routes>
                <Route path="/" element={<PayoutForm />} />
                <Route path="/success" element={<SuccessPage />} />
            </Routes>
        </Router>
    );
};
```

```
export default App;
```

PayoutForm.js

The `PayoutForm.js` component collects the necessary data from the merchant and sends it to the backend API. Upon successful submission, it redirects the user to the success page.

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';

const PayoutForm = () => {
    const [formData, setFormData] = useState({
        merchantId: '',
        merchantOrderId: '',
        amount: '',
        phone: '',
        email: '',
        account: '',
        accountName: '',
        address: '',
        subBranch: '',
        withdrawType: 1,
        bankName: '',
        remark: '',
        tunnelId: '',
        currency: 'USD',
        nonce: '',
        timestamp: ''
    });

    const navigate = useNavigate();

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData({ ...formData, [name]: value });
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        formData["timestamp"] = Date.now();
        try {
```

```
            const response = await
axios.post('http://localhost:5000/payment/payout', formData);
            if (response.data.code === 200) {
                navigate('/success', { state: { data: response.data.data } });
            }
        } catch (error) {
            console.error('Error submitting payout form:', error);
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <input type="text" name="merchantId" placeholder="Merchant ID"
value={formData.merchantId} onChange={handleChange} required />
            <input type="text" name="merchantOrderId" placeholder="Merchant Order
ID" value={formData.merchantOrderId} onChange={handleChange} required />
            <input type="text" name="amount" placeholder="Amount"
value={formData.amount} onChange={handleChange} required />
            <input type="text" name="phone" placeholder="Phone"
value={formData.phone} onChange={handleChange} required />
            <input type="email" name="email" placeholder="Email"
value={formData.email} onChange={handleChange} required />
            <input type="text" name="account" placeholder="Account"
value={formData.account} onChange={handleChange} required />
            <input type="text" name="accountName" placeholder="Account Name"
value={formData.accountName} onChange={handleChange} required />
            <input type="text" name="address" placeholder="Address"
value={formData.address} onChange={handleChange} required />
            <input type="text" name="subBranch" placeholder="Sub Branch"
value={formData.subBranch} onChange={handleChange} required />
            <input type="text" name="withdrawType" placeholder="Withdraw Type"
value={formData.withdrawType} onChange={handleChange} required />
            <input type="text" name="bankName" placeholder="Bank Name"
value={formData.bankName} onChange={handleChange} required />
            <input type="text" name="remark" placeholder="Remark"
value={formData.remark} onChange={handleChange} />
            <input type="text" name="tunnelId" placeholder="Tunnel ID"
value={formData.tunnelId} onChange={handleChange} required />
            <input type="text" name="currency" placeholder="Currency"
value={formData.currency} onChange={handleChange} required />
            <input type="text" name="nonce" placeholder="Nonce"
value={formData.nonce} onChange={handleChange} required />
            <button type="submit">Submit</button>
        </form>
    );
```

```
};

export default PayoutForm;
```

`SuccessPage.js`

The `SuccessPage.js` component displays the result of a successful payment transaction.

```
import React from 'react';
import { useLocation } from 'react-router-dom';

const SuccessPage = () => {
    const location = useLocation();
    const { data } = location.state;

    return (
        <div>
            <h1>Payment Successful</h1>
            <p>Merchant ID: {data.merchantId}</p>
            <p>Timestamp: {data.timestamp}</p>
        </div>
    );
};

export default SuccessPage;
```

Project Workflow and Data Flow

The project integrates ZipPay's payment gateway using Node.js and React, providing a complete solution for handling payments from a frontend form to backend processing and secure communication. This section explains the entire workflow, including how communication and data flow occur within the system.

Overview

1. **Frontend**: A React application where users (merchants) input payment details.
2. **Backend**: A Node.js server that processes payment requests, verifies signatures, and communicates with the mock ZipPay API.
3. **Mock Server**: Simulates ZipPay's API for testing purposes, handling payment requests and sending callbacks.

Detailed Explanation

Frontend (React)

1. **User Input**:
    - The merchant enters payment details in a form provided by the `PayoutForm` component.
    - The form data includes merchant ID, order ID, amount, contact details, account information, and more.
2. **Form Submission**:
    - When the merchant submits the form, the data is bundled into a JSON object.
    - An HTTP POST request is made to the backend endpoint (`/payment/payout`) using Axios.

```
const handleSubmit = async (e) => {
    e.preventDefault();
    try {
        const response = await
axios.post('http://localhost:5000/payment/payout', formData);
        if (response.data.code === 200) {
            navigate('/success', { state: { data: response.data.data } });
        }
    } catch (error) {
        console.error('Error submitting payout form:', error);
    }
};
```

3. **Success Page**:

- If the backend confirms the payment, the merchant is redirected to the `SuccessPage`.
- The success page displays the payment details received from the backend.

```
const SuccessPage = () => {
    const location = useLocation();
    const { data } = location.state;

    return (
        <div>
            <h1>Payment Successful</h1>
            <p>Merchant ID: {data.merchantId}</p>
            <p>Timestamp: {data.timestamp}</p>
        </div>
    );
};
```

Backend (Node.js)

1. **Receiving Payment Request**:
    - The backend endpoint `/payment/payout` receives the JSON object from the frontend.
    - It extracts the necessary fields and constructs a signature string for verification.

```
app.post('/payment/payout', (req, res) => {
    const payLoad = req.body;
    const signData = payLoad.merchantId + payLoad.merchantOrderId +
payLoad.amount + payLoad.nonce + payLoad.timestamp;
    const isValidSign = cleanUpDecryptedData(decryptData(payLoad.sign)) ==
signData;
```

2. **Signature Verification**:

- The backend verifies the signature to ensure data integrity and authenticity.
- If the signature is valid, it simulates payment processing.

3. **Payment Processing**:

- The backend simulates the payment process and determines the transaction outcome.
- If successful, it constructs a callback payload and sends it to the mock server.

```
if (isValidSign) {
    let isPaymentSuccessful = true; // Simulated outcome
    let data = {
        code: 1,
        msg: 'SUCCESS',
        merchantOrderId: payLoad.merchantOrderId,
        platformOrderId: "A22308231907420000184",
        utr: ''
    };
    Object.keys(data).forEach(key => {
        if (data[key] === undefined || data[key] === null || data[key] ===
'') {
            delete data[key];
        }
    });
    const sign = encryptData(constructSignData(data));
    data["sign"] = sign;
```

```
    axios.post('http://localhost:5000/payment/payout/callback', data);
```

4. **Sending Response**:

- The backend sends a response back to the frontend, confirming the transaction outcome.

```
res.status(200).json({
    code: 200,
    msg: 'SUCCESS',
    success: true,
    data: {
        merchantId: payLoad.merchantId,
        merchanOrderId: payLoad.merchanOrderId,
        platformOrderId: payLoad.platformOrderId,
        timestamp: `${Date.now()}`
    }
});
```

5. **Handling Callbacks**:

- The backend listens for callback requests from the mock server.
- It verifies the callback data and processes it accordingly.

```
app.post('/payment/payout/callback', (req, res) => {
    const { code, msg, merchantOrderId, platformOrderId, amount, realAmount,
sign, utr } = req.body;
    const data = { code, msg, merchantOrderId, platformOrderId, amount,
realAmount, utr };
    Object.keys(data).forEach(key => {
        if (data[key] === undefined || data[key] === null || data[key] ===
'') {
            delete data[key];
        }
    });
    const signData = constructSignData(data);
    const isValidSign = decryptData(sign) === signData;
    if (isValidSign) {
        console.log('Callback data processed successfully');
        res.send('success');
    } else {
        res.status(400).send('invalid signature');
    }
});
```

Mock Server (Node.js)

1. **Receiving Payment Request**:
    - The mock server receives the payment request from the backend.
    - It verifies the signature and simulates the payment process.
2. **Sending Callback**:
    - The mock server constructs a callback payload with the transaction result.
    - It sends the callback to the backend's callback endpoint.

```
app.post('/payment/payout', (req, res) => {
    const payLoad = req.body;
    const signData = payLoad.merchantId + payLoad.merchantOrderId +
payLoad.amount + payLoad.nonce + payLoad.timestamp;
    const isValidSign = cleanUpDecryptedData(decryptData(payLoad.sign)) ==
signData;
    if (isValidSign) {
        let isPaymentSuccessful = true;
        let data = {
            code: 1,
            msg: 'SUCCESS',
            merchantOrderId: payLoad.merchantOrderId,
            platformOrderId: "A22308231907420000184",
            utr: ''
        };
        Object.keys(data).forEach(key => {
            if (data[key] === undefined || data[key] === null || data[key]
=== '') {
                delete data[key];
            }
        });
        const sign = encryptData(constructSignData(data));
        data["sign"] = sign;
        axios.post('http://localhost:5000/payment/payout/callback', data)
            .then((response) => {
                console.log('Payout callback sent successfully');
            }).catch((err) => {
                console.error('Error sending callback:', err.message);
            });
        res.status(200).json({
            code: 200,
            msg: 'SUCCESS',
            success: true,
            data: {
                merchantId: payLoad.merchantId,
                merchanOrderId: payLoad.merchanOrderId,
```

```
                platformOrderId: payLoad.platformOrderId,
                timestamp: `${Date.now()}`
            }
        });
        console.log("Payout was successful");
    } else {
        res.status(400).send('invalid signature');
    }
});
```

Communication and Data Flow

1. **Form Submission**: The merchant fills out the payment form on the frontend and submits it.
2. **Data Transmission**: The form data is sent to the backend via an HTTP POST request.
3. **Signature Verification**: The backend verifies the signature to ensure the data's integrity.
4. **Payment Processing**: The backend processes the payment and sends a callback to the mock server.
5. **Callback Handling**: The mock server sends a callback response to the backend.
6. **Response to Frontend**: The backend sends a response back to the frontend, and the user is redirected to the success page.

This workflow ensures secure, seamless, and efficient handling of payment transactions between the frontend, backend, and ZipPay's simulated API.

Conclusion

This implementation covers the integration of ZipPay's payment gateway using Node.js for the backend and React for the frontend. The backend ensures secure communication through encryption and decryption of transaction data, while the frontend provides an interface for merchants to initiate payments and view the results. The use of a mock server for testing allows for safe and efficient development and debugging.