API Authentication

To authenticate with the DuitNow Pay API, JSON Web Token (JWT) is required. The JWT token should be included in the "Authorization" request header. Acquirer is responsible to generate an RS256 key pair and sharing the public key with PayNet during the onboarding process. Prior to making a call to the DuitNow Pay API, acquirer needs to generate a token by signing the payload with the private key, together with specific JWT claims, each serving a crucial role in the authentication process.

Claims	Value
exp (expiration time)	Expiry date time (refer to sample generator below).
iss (issuer)	Client Id, or acquirer Id assigned during onboarding process.
sub (subject)	Merchant Id assigned during onboarding process.
jti (JWT id)	UUID v4 generated by merchant.
key (custom claim)	Merchant key name assigned during onboarding process.
data (custom claim)	Request payload.

Paynet will share public key so that acquirer can verify API response from DuitNow Pay.

In the following generator example,

1. Private key value should be set with **acquirer's private key**, to sign API request to Paynet.
2. Public key value should be set with **Paynet's public key**, to verify API response from Paynet.

Sample Javascript Token Generator

```
const jwt = require('jsonwebtoken');
const uuid = require('uuid');
const NodeRSA = require('node-rsa');

const apiUUID = uuid.v4();

// Sample payload
const bodyJson =
{
  checkoutId: apiUUID,
  amount: '10.00',
  merchant: { productId: 'P00000205'},
  merchantReferenceId: 'ref20240124T073716',
  customer: {
      name: 'Walter Mitty',
      identityValidation: '00',
      identificationType: '05',
      identification: '+60123456789',
  },
  consent: {
      maxAmount: '100.00',
      effectiveDate: '2024-01-24',
      expiryDate: '2024-04-24',
      frequency: '01',
```

```
  }
};
const payload = JSON.stringify(bodyJson);

// Sample private key
const privateKeyString = `-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDJ5qp0PdVBXe7L
... ...
Ztbkr3gEJFQNe2cC/p0zDju7xF/8J1SxtC1/L30nrzyNpbPhjdEPNayrZnzsCSRN
SWWqUtIVB+D60rbYq6UW7YxC
-----END PRIVATE KEY-----`;

// Sample public key
const publicKeyString = `-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyeaqdD3VQV3uy5jAXZHF
... ...
Y79wIaebDvANPJUaoXqdr8lmGYhbFLA1IeNAmIHCphv5pMayftmi+g2JcXF8B3W9
WQIDAQAB
-----END PUBLIC KEY-----`;

// Load private and public keys
const privateKey = new NodeRSA(privateKeyString);
const publicKey = new NodeRSA(publicKeyString);

// Sign the payload
const token = jwt.sign({ data: bodyJson, key: '<api-key-name>' },
privateKey.exportKey('pkcs8-private'), {
  algorithm: 'RS256',
  expiresIn: '1h', // Token valid for 1 hour
  issuer: '<acquirerId>',
  subject: '<merchantId>',
```

```
  jwtid: apiUUID,
});

console.log('Generated Token:', token);

// Verify the token
jwt.verify(token, publicKey.exportKey('pkcs8-public'), (err, decoded) => {
  if (err) {
    console.error('Signature verification failed:', err.message);
    return;
  }

  console.log('Signature verified. Decoded Token:', decoded);
});
```

Java Token Generator

```
package my.paynet.duitnowpay;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.nimbusds.jose.JWSAlgorithm;
import com.nimbusds.jose.JWSHeader;
import com.nimbusds.jose.crypto.RSASSASigner;
import com.nimbusds.jose.crypto.RSASSAVerifier;
import com.nimbusds.jwt.JWTClaimsSet;
import com.nimbusds.jwt.SignedJWT;
```

```
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import java.util.Date;
import java.util.UUID;

public class Main {

    private static PrivateKey loadPrivateKey(String privateKeyString) throws
Exception {
        String privateKeyPEM = privateKeyString
                .replace("-----BEGIN PRIVATE KEY-----", "")
                .replace("-----END PRIVATE KEY-----", "")
                .replaceAll("\\s", "");
        return KeyFactory.getInstance("RSA")
                .generatePrivate(
                        new
PKCS8EncodedKeySpec(Base64.getDecoder().decode(privateKeyPEM)));
    }

    public static PublicKey loadPublicKey(String publicKeyString) throws
Exception {
        String publicKeyPEM = publicKeyString
                .replace("-----BEGIN PUBLIC KEY-----", "")
                .replace("-----END PUBLIC KEY-----", "")
                .replaceAll("\\s+", "");
        return KeyFactory.getInstance("RSA")
```

```
                .generatePublic(new
X509EncodedKeySpec(java.util.Base64.getDecoder().decode(publicKeyPEM)));
    }

    private static Object convertPayloadToCheckOutRequest(String payload) {
        ObjectMapper objectMapper = new ObjectMapper();

objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
false);
        try {
            return objectMapper.readValue(payload, Object.class);
        } catch (Exception e) {
            e.printStackTrace();
            return new Object();
        }
    }

    private static void printAllClaims(JWTClaimsSet claimsSet) throws
Exception {
        System.out.println("List of Claims:");
        System.out.println("Subject: " + claimsSet.getSubject());
        System.out.println("Issuer: " + claimsSet.getIssuer());
        System.out.println("Expiration Time: " +
claimsSet.getExpirationTime());
        System.out.println("Issue Time: " + claimsSet.getIssueTime());
        claimsSet.getClaims().forEach((key, value) ->
                System.out.println("Custom Claim - " + key + ": " + value)
        );
    }

    public static void main(String[] args) throws Exception {
```

```
        var uuid = UUID.randomUUID().toString();
        String bodyJson = "
{\"consentDa\":\"Maybank\",\"amount\":\"10.00\",\"merchant\":
{\"productId\":\"P00000205\"}" +
                ",\"sourceOfFunds\":[\"01\"],\"checkoutId\":\"" + uuid +
"\",\"merchantReferenceId\":\"REF0012345678\"" +
                ",\"consent\":{\"expiryDate\":\"2024-04-
12\",\"allowTerminatedByDebtor\":\"false\"" +
                ",\"maxAmount\":\"100.00\",\"effectiveDate\":\"2024-01-
12\",\"frequency\":\"01\"}" +
                ",\"customer\":
{\"identification\":\"+60123456789\",\"name\":\"Walter Mitty\"" +

",\"identificationType\":\"05\"}}";System.out.println("bodyJson " +
bodyJson);
        Object payload = convertPayloadToCheckOutRequest(bodyJson);

        //consumer def_jwt_auth private key
        String privateKeyString = "-----BEGIN PRIVATE KEY-----\n" +

"MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDJ5qp0PdVBXe7L\n" +
                "... ...\n" +

"Ztbkr3gEJFQNe2cC/p0zDju7xF/8J1SxtC1/L30nrzyNpbPhjdEPNayrZnzsCSRN\n" +
                "SWWqUtIVB+D60rbYq6UW7YxC\n" +
                "-----END PRIVATE KEY-----";

        // Your public key string (for verification)
        String publicKeyString = "-----BEGIN PUBLIC KEY-----\n" +

"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyeaqdD3VQV3uy5jAXZHF\n" +
```

```
                "... ...\n" +

"Y79wIaebDvANPJUaoXqdr8lmGYhbFLA1IeNAmIHCphv5pMayftmi+g2JcXF8B3W9\n" +
                "WQIDAQAB\n" +
                "-----END PUBLIC KEY-----";

        var currDateTime = new Date().getTime();
        JWTClaimsSet claimsSet = new JWTClaimsSet.Builder()
                .subject("<merchantId>")
                .issuer("<acquirerId>")
                .issueTime(new Date(currDateTime))
                .expirationTime(new Date(currDateTime + 60 * 60000)) // Token
valid for 60 minutes
                .jwtID(uuid)
                .claim("key", "<api-key-name>") /* REMEMBER change key name
if use other key */
                .claim("data", payload)
                .build();

        printAllClaims(claimsSet);

        SignedJWT signedJWT = new SignedJWT(new
JWSHeader.Builder(JWSAlgorithm.RS256).build(), claimsSet);
        signedJWT.sign(new RSASSASigner(loadPrivateKey(privateKeyString)));
        String jwtString = signedJWT.serialize();

        System.out.println("Generated Token: " + jwtString);

        boolean isValid = SignedJWT.parse(jwtString)
                .verify(new RSASSAVerifier((RSAPublicKey)
loadPublicKey(publicKeyString)));
```

```
        System.out.println("Is signature valid? " + isValid);
    }
}
```

Python Token Generator

```
# pip install PyJWT
# pip install cryptography
# pip install pycryptodome

import jwt
import uuid
from Crypto.PublicKey import RSA
from datetime import datetime, timedelta

api_uuid = str(uuid.uuid4())

# Sample payload
body_json = {
    'checkoutId': api_uuid,
    'amount': '10.00',
    'merchant': {'productId': 'P00000501'},
    'merchantReferenceId': 'ref20240124T073716',
    'customer': {
        'name': 'Walter Mitty',
        'identityValidation': '00',
        'identificationType': '05',
```

```
        'identification': '+60123456789',
    },
    'consent': {
        'maxAmount': '100.00',
        'effectiveDate': '2024-01-24',
        'expiryDate': '2024-04-24',
        'frequency': '01',
    }
}

# Sample private key
private_key_string = """-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDJ5qp0PdVBXe7L
... ...
Ztbkr3gEJFQNe2cC/p0zDju7xF/8J1SxtC1/L30nrzyNpbPhjdEPNayrZnzsCSRN
SWWqUtIVB+D60rbYq6UW7YxC
-----END PRIVATE KEY-----"""

# Load private key and convert to PEM format
private_key = RSA.import_key(private_key_string)
private_key_pem = private_key.export_key(format='PEM')

# Sign the payload
token = jwt.encode({'data': body_json, 'key': '<api-key-name>', 'jti':
api_uuid, 'iat': datetime.utcnow(), 'exp': datetime.utcnow() +
timedelta(hours=1), 'sub': '<merchantId>', 'iss': '<acquirerId>'},
private_key_pem, algorithm='RS256')

print('Generated Token:', token)
```