Orange API Documentation

Recurring Payment API v2.0.0

Abstract

These APIs are designed to enable recurring payment functionalities that support customer subscriptions for merchants and businesses offering dynamic plans, facilitating seamless and automated financial interactions across platforms.

Orange Money Technical Team

orangemoneytechnical.ojo@orange.com

11 November 2025

Orange API Documentation

This document contains a collection of APIs that serve Recurring Payments functionalities that serve customer subscriptions on merchants' platforms and business entities with dynamic plans.

1. Introduction

These APIs are intended for partners and developers integrating wallet services into their platforms for enhanced financial interactions between individual users and businesses.

Base URL (Production): https://orangemoney.orange.jo:1594/

Base URL (Sandbox): https://om-dev.orange.jo:1445/

2. Test requirement

To ensure a smooth integration and testing processes. The following points need to be considered for testing and production:

1. **Two phone numbers are required for testing purposes:**
   - **One associated with the Personal Wallet.**
   - **One associated with the Business/Merchant Wallet.**
2. **Credentials will be sent by Orange Money through text message on the number related to business wallet.**
3. **Credentials for testing (staging) are different from the ones for live (production).**
4. **The agent needs to provide a static IP address that needs to be whitelisted by Orange Money during testing (staging) and another one for live (production).**
5. **The agent can review all transactions from the Agent Portal.**

3. Authentication

All API endpoints require JWT token–based authentication to be included in the Authorization header:

- **Authorization: Bearer <AccessToken>.**

📌 ***Notes:***

- The external client must first authenticate using the Authorization API, which returns a JWT token.

Authorization

Path	api/ExternalAPI/Authorization
HTTP Method	POST
JSON format	Content-Type: application/json

Request

Parameter	Type	Required	Description	Encryption
UserName	string	Yes	The UserName	Yes
Password	string	Yes	The Password	Yes

Response

Parameter	Type	Description
AccessToken	String	Access Token
token_type	String	Toke type always Bearer
expires_in	int	ExpiresIn
errorCode	String	Error code
errorDescription	String	Error message text
isSuccess	bool	Indicates if the transaction was successful
IsOTPRequired	bool	Specifies if an OTP is required for the transaction
errors	List [{ "description": "string", "descriptionAr": "string", }]	List of Errors

4. API Structure & Design

Recurring Payment

Path	**api/Subscriptions/V2/RecurringPayment**
HTTP Method	**POST**
JSON format	**Content-Type: application/json**

5. Error Handling

The API's use standard HTTP status codes:

Code	Meaning
200	**OK**
500	**Internal Server Error**

The API's also return the following Application-Level errors codes:

📌 ***Note: these error codes are returned in the response with 200 OK HTTP status code.***

Code	Description	Description Arabic
236	The amount entered is invalid	المبلغ المدخل غير صحيح
311	**The minimum allowed amount is 0.1 JD**	الحد الأدنى للمبلغ المسموح به هو **0.1** دينار
53	Transaction is not allowed	المعاملة غير مسموح بها
230	Invalid Signature	Invalid Signature
110	Agent wallet not found	محفظة الوكيل غير موجودة
334	The user is not authorized to perform this transaction	المستخدم غير مخول بإجراء هذه المعاملة
335	Duplicate Merchant Reference	رقم المرجع مكرر

46	Transaction amount exceeds the allowed limit	قيمة هذه الحركة أعلى من الحد المسموح
112	Sorry, we are facing an issue at the moment, please try again later	نأسف، نواجه مشكلة في الوقت الحالي، يرجى المحاولة لاحقاً
187	Invalid Transaction	معاملة غير صحيحة
407	Invalid Reference Number	رقم مرجعي غير صالح
411	Duplicate Merchant Transaction Reference	مرجع معاملة التاجر المكرر
406	Invalid Merchant Code	رمز التاجر غير صالح

6. Endpoints

6.1 Recurring Payment

Method	**POST**
URL	**api/Subscriptions/V2/RecurringPayment**
Header	**Key: Signature** **Value: "string"**
Request	**{** **"MerchantCode": "U2FsdRVJDSEFOVF9DT0RFXzEyMw==",** **"ReferenaceNumber ": "U2FsdGVkX1VTVE9NRVJfOTg3NjU=",** **"CustomerUID": "U2FsdGVk5BTElEXzEyMzQ1Njc4OTA=",** **"Amount": "U2FsdGVkX19QUkVNSVVNXzUwMC4wMA==",** **"MerchantTransactionRef": "U2FsdG9UWE5fMjAyNTExMTBfM==",** **"NoOfInstallments": "U2FsdGVkjjrt5956X19JTlNfMT= =",** **"TransactionNote": "U2FsdGVkX19NByZW1pdW1fcGbWVudA==",** **"RecurringNote": "U2FsdGVkXVic2NyaXB0aW9uX3BsYW4="** **}**

Response	{ "TransactionTypeName": "xxxxxxx", "SenderName": "xxxxxx", "FeesWalletName": "OJM FEES", "SenderAgentId": 2476, "ReceiverAgentId": 1, "SenderWalletId": 33197, "ReceiverWalletId": 33199, "FeesWalletId": 5460, "SenderClientTypeId": 3, "ReceiverClientTypeId": 1, "SenderWalletTypeId": 735, "ReceiverWalletTypeId": 54, "SenderCorperateName": "AlBasheqco", "ReceiverCorperateName": "", "FeesWalletCorperateName": "OJMFEES", "SenderBalanceBefore": 10004.95, "ReceiverBalanceBefore": 495, "FeesWalletBalanceBefore": 31817.463, "SenderBalanceAfter": 9999.7, "ReceiverBalanceAfter": 500, "FeesWalletBalanceAfter": 31817.713, "TotalFeesAmount": 0, "TotalTransactionAmount": 5, "OtherTotalTransactionAmount": 5.25, "OtherTotalFeesAmount": 0.25, "SumOfTotalTransactionAmount": 5.25, "SumOfTotalFeesAmount": 0.25, "TransactionDateTime": "11/04/2024 16:31:24", "AgentName": "AlBasheq", "IsOTPGenerated": false, "SenderWallet": null,

"ReceiverWallet": null,
"ErrorMessage": "",
"BulkExternalFees": 0,
"ThirdPartyFees": 0,
"FromAccount": "ORNG-CNIDL0798898691W33197",
"ToAccount": "ORNG-C2024101313W33199",
"CustomerRecordId": "133444",
"IsOffUs": null,
"SenderAddress": "Jordan Amman",
"ReceiverAddress": "Jordan Amman",
"CommercialRegisterName": "",
"ReceiverCustomerRecordId": null,
"ReceiverAccountRecordId": null,
"ReceiverAccountNumber": null,
"ServicerName": null,
"ServicerNameAr": null,
"SenderReceiverName": null,
"ReceiverFeesValue": 0,
"RechargePin": null,
"ExpiryDate": null,
"Serial": null,
"WalletUserName": null,
"MSISDN": null,
"TransactionAmount": 5,
"TransactionFees": 0,
"TransactionTotalAmount": 0,
"WalletBalanceBefore": 0,
"WalletBalanceAfter": 0,
"TransactionId": 2850475,
"OJPaymentId": null,
"transactionReference": "OM20241104163124720343259701 5",
"ReceiverName": "basheq Test basheq Test",

```
"ServicerNameEn": null,
"SenderPhonerNumber": "0798898691",
"ReceiverPhonerNumber": "0795968391",
"FeesWalletPhonerNumber": "0798528839",
"isSuccess": true,
"IsOTPRequired": false,
"errors": []
}
```


📌 *Note: The official reference for the transaction is TransactionReference. This should be used as the primary identifier for tracking any transaction.*

7. Data Models

7.1 Recurring Payment

7.1.1 Request

Field	Type	Required	Encrypted	Description
MerchantCode	string	Yes	Yes	Code of the user (will be provided soon from orange)
ReferenaceNumber	string	Yes	Yes	reference to Identify the Customer (generated from OM side)
CustomerUID	string	Yes	Yes	reference to Identify the Customer (generated from Merchant side, Metlife should send NationalID)
Amount	string	Yes	Yes	the amount of transaction
MerchantTransactionRef	string	Yes	Yes	unique transaction reference

				(generated from merchant)
NoOfInstallments	int	No	No	Number of installments that requested from customer
TransactionNote	string	No	Yes	
RecurringNote	string	No	Yes	Any note you want

7.1.2 Response

Field	Type	Description
TransactionTypeName	String	Name or label identifying the type of transaction (e.g., Money Transfer, Bill Payment).
SenderName	String	Name of the person or entity initiating the transaction.
FeesWalletName	String	Name of the wallet where applicable fees are charged.
SenderAgentId	Integer	Unique identifier of the agent initiating the transaction on behalf of the sender
ReceiverAgentId	Integer	Unique identifier of the agent associated with the receiver
SenderWalletId	Integer	Unique identifier of the sender's digital wallet.
ReceiverWalletId	Integer	Unique identifier of the receiver's digital wallet
FeesWalletId	Integer	Unique identifier of the wallet used to collect transaction fees
SenderClientTypeId	Integer	Type ID representing the sender's client category (e.g., Individual, Business).
ReceiverClientTypeId	Integer	Type ID representing the receiver's client category
SenderWalletTypeId	Integer	ID representing the type of sender's wallet (e.g., Personal, Merchant)
ReceiverWalletTypeId	Integer	ID representing the type of receiver's wallet
SenderCorperateName	String	Sender Corporate Name

ReceiverCorperateName	String	Receiver Corporate Name
FeesWalletCorperateName	String	Fees Wallet Corporate Name
SenderBalanceBefore	Decimal	Wallet balance of the sender before executing the transaction
ReceiverBalanceBefore	Decimal	Wallet balance of the receiver before executing the transaction
FeesWalletBalanceBefore	Decimal	Balance of the fees wallet before transaction charges are applied.
SenderBalanceAfter	Decimal	Wallet balance of the sender after the transaction is completed.
ReceiverBalanceAfter	Decimal	Wallet balance of the receiver after the transaction is completed.
FeesWalletBalanceAfter	Decimal	Balance of the fees wallet after transaction charges are deducted.
TotalFeesAmount	Decimal	Total amount of fees charged for the transaction
TotalTransactionAmount	Decimal	Actual transaction amount excluding fees.
OtherTotalTransactionAmount	Decimal	Any additional transaction amounts (e.g., charges in other currencies).
OtherTotalFeesAmount	Decimal	Additional or external fee components
SumOfTotalTransactionAmount	Decimal	Cumulative total of all transaction amounts within a batch or summary.
SumOfTotalFeesAmount	Decimal	Cumulative total of all fees charged across transactions
TransactionDateTime	DateTime	Exact date and time when the transaction was performed (e.g., "11/04/2024 16:31:24").
AgentName	String	Name of the agent associated with the transaction.

IsOTPGenerated	Boolean	Indicates whether an OTP was generated for the transaction process.
SenderWallet	Object	Detailed object containing the sender's wallet information.
ReceiverWallet	Object	Detailed object containing the receiver's wallet information.
ErrorMessage	String	Error message returned by the system in case of transaction failure.
BulkExternalFees	Decimal	External fees applicable in the context of bulk transactions.
ThirdPartyFees	Decimal	Fees charged by third parties Fees imposed by third-party services or payment gateways.
FromAccount	String	The source account number for the transaction (e.g., "ORNG-CNIDL0798898691W33197")
ToAccount	String	The destination account number for the transaction (e.g., "ORNG-C2024101313W33199")
CustomerRecordId	String	Unique customer identifier used in systems like CliQ.
IsOffUs	Boolean	Indicates if the transaction is "off us"
SenderAddress	String	The address of the sender (e.g., "Jordan Amman")
ReceiverAddress	String	The address of the receiver (e.g., "Jordan Amman")
CommercialRegisterName	String	Official commercial name as per the commercial registry.
ReceiverCustomerRecordId	String	Unique record ID identifying the receiver in CliQ or other systems.
ReceiverAccountRecordId	String	The account record ID for the receiver
ReceiverAccountNumber	String	The account number of the receiver

ServicerName	String	The name of the service provider
ServicerNameAr	String	The service provider's name in Arabic
SenderReceiverName	String	The sender-receiver's name
ReceiverFeesValue	Decimal	The value of any fees charged to the receiver
RechargePin	String	Recharge or voucher PIN (used in recharge services).
ExpiryDate	DateTime	The expiry date of the transaction or service
Serial	String	Unique serial number related to the transaction or item.
WalletUserName	String	The username associated with the wallet
MSISDN	String	Mobile number (MSISDN) associated with the transaction.
TransactionAmount	Decimal	The core amount to be transferred in the transaction
TransactionFees	Decimal	The fee amount applied to this specific transaction.
TransactionTotalAmount	Decimal	The total amount including fees
WalletBalanceBefore	Decimal	Wallet balance before initiating the transaction
WalletBalanceAfter	Decimal	Wallet balance after completing the transaction
TransactionId	Integer	Unique numeric identifier assigned to the transaction.
OJPaymentId	String	The payment ID for the transaction
TransactionReference	String	The reference ID for the transaction (e.g., "OM20241104163124720343259 7015").
ReceiverName	String	The name of the receiver
ServicerNameEn	String	The name of the service
SenderPhoneNumber	String	The sender's phone number

ReceiverPhoneNumber	String	The receiver's phone number
FeesWalletPhoneNumber	String	The fees wallet phone number
isSuccess	bool	Indicates if the transaction was successful
IsOTPRequired	bool	Specifies if an OTP is required for the transaction
errors	List [{ "description": "string", "descriptionAr": "string", }]	List of Errors

8. Security

To ensure secure communication and data integrity between client systems and the API's, the followings must be followed:

8.1 Encryption:

The encryption algorithm used is Advanced Encryption Standard (AES) with the flowing AES Key & HMAC Key:

Recurring Payment

	Staging	Production
AES Key	hzwFdEEXY+j9z13s7X10gU/MgCBQGlAHR2rFbVIL6bc=	eZ1l1LaTz56R40CYf4bt012P/wv1xm7lQVmpKHRk8dY=
HMAC Key	F99dqrMqL23ukgKkURrj7TcQ79tL2ze66p1gFFJaJjY=	Y0nC0dFxn2tCTK1BoboZuKmxQwUZLMaiuaEldMO/4k8=

Here's a step-by-step breakdown of how the method Encrypt AES works:

1. Decoding Input Keys (AES and HMAC):

- AES Key:
  The provided base64AesKey string is decoded into a byte array to be used for the AES encryption process.

- **HMAC Key:**
  The base64HmacKey string is also decoded into a byte array, which will be used to compute the HMAC for integrity verification.

```
C# Example:
byte[] key = Convert.FromBase64String(AesKey);
byte[] hmacKey =
Convert.FromBase64String(HmacKey);
```

2. Setting Up AES for Encryption:

- **An Aes object is created and configured with the provided AES key.**
- **Mode: Set to CBC (Cipher Block Chaining), which requires an Initialization Vector (IV) for encryption.**
- **Padding: Set to PKCS7 (to handle cases where the plaintext size isn't a multiple of the AES block size).**
- **IV Generation: A random IV is generated for this encryption session to ensure that even if the same plaintext is encrypted multiple times, the ciphertext will be different each time.**

```
C# Example:
using var aes = Aes.Create();
aes.Key = key;
aes.Mode = CipherMode.CBC;
aes.Padding = PaddingMode.PKCS7;
aes.GenerateIV(); // Generates a new random IV each time
```

3. Encrypting the Plaintext:

- **Encryptor Creation: A CryptoStream is used to perform the actual encryption. It applies the AES encryption to the plaintext and stores the result in the cipherBytes array.**
- **The plaintext is written to a StreamWriter, which is wrapped in a CryptoStream. The CryptoStream applies the encryption and writes the encrypted data to a MemoryStream.**
- **Ciphertext: The result is the encrypted data (ciphertext) of the plaintext.**

```
C# Example:
byte[] cipherBytes;
using (var encryptor = aes.CreateEncryptor())
```

```
using (var ms = new MemoryStream())
using (var cs = new CryptoStream(ms, encryptor,
CryptoStreamMode.Write))
using (var sw = new StreamWriter(cs, Encoding.UTF8))
{
    sw.Write(plainText);
    sw.Close();
    cipherBytes = ms.ToArray();
}
```

4. Combining IV and Ciphertext:

- The IV (Initialization Vector) and the ciphertext are concatenated together into a single byte array (ivAndCipher). This step ensures that the IV is included in the final encrypted data.
- IV: The IV is placed at the beginning, followed by the ciphertext.

```
C# Example:
byte[] ivAndCipher = new byte[aes.IV.Length + cipherBytes.Length];
Buffer.BlockCopy(aes.IV, 0, ivAndCipher, 0, aes.IV.Length);
Buffer.BlockCopy(cipherBytes, 0, ivAndCipher, aes.IV.Length,
cipherBytes.Length);
```

5. Computing the HMAC for Integrity:

- HMAC Calculation: A HMAC-SHA256 hash is calculated over the concatenated IV and ciphertext (ivAndCipher). This HMAC ensures that the encrypted data has not been tampered with.
- HMAC Key: The HMAC is computed using the provided HMAC key.

```
C# Example:
byte[] hmac;
using (var hmacSha = new HMACSHA256(hmacKey))
hmac = hmacSha.ComputeHash(ivAndCipher);
```

6. Combining HMAC, IV, and Ciphertext:

- The HMAC, IV, and Ciphertext are concatenated together into a final byte array (finalData).
- This final byte array represents the complete encrypted data that includes both the ciphertext and the HMAC for integrity checking.

```
C# Example:
```

```
byte[] finalData = new byte[hmac.Length + ivAndCipher.Length];
Buffer.BlockCopy(hmac, 0, finalData, 0, hmac.Length);
Buffer.BlockCopy(ivAndCipher, 0, finalData, hmac.Length,
ivAndCipher.Length);
```

7. Base64 Encoding:

- The final concatenated data (HMAC | IV | Ciphertext) is base64-encoded to make it suitable for transmission or storage. This ensures that the resulting string is in a safe, readable format.
- The base64-encoded result is returned as a string.

```
C# Example:
return Convert.ToBase64String(finalData);
```

8.2 Signature From Header :

The signature field is used to validate the authenticity of the request. The signature is generated using the SHA-256 hashing algorithm.

📌 *Note: use this link https://emn178.github.io/online-tools/sha256.html*

To generate the signature, use the following formulas per each API:

	Signature Formula
Recurring Payment	(API KEY + MerchantCode + CustomerUID + Amount + API KEY)

API KEY: ABC123

📌 *Note: All the above signature values must be generated before hashing and/or encryption.*

9. Contact Information

For support or questions, please contact:

Email: orangemoneytechnical.ojo@orange.com

Phone: