# Full API Integration

Full API Integration is designed for partners who want to embed the full card issuing flow directly into their own system.

In this model, the partner does not use Sparados hosted mobile or web applications. Instead, the partner manages the end-user experience on its own side and communicates with Sparados APIs directly.

Full API Integration allows partners to:

- create users via API
- create cards via API
- manage card lifecycle
- retrieve and display card data, including PAN / CVV
- handle OTP and 3DS flows
- receive Apple Pay / Google Pay OTP notifications
- retrieve transactions and receive transaction notifications

This integration model gives the partner full control over the user and card flow, but may require PCI DSS compliance if sensitive card data is received, processed, stored, or displayed.

Before using Full API Integration, the partner must complete onboarding with Sparados, sign the cooperation agreement, and configure Mutual TLS authentication for the selected environment.

**API Addresses**

| API | Integration | BETA | PROD |
|---|---|---|---|
| Cards API | Simple / Full | `https://sparadosv2-bc-api.secure-verestro.dev/secure` | `https://sparadosv2-bc-api.secure-verestro.com/secure` |
| Transaction API | Simple / Full | `https://services.upaidtest.pl/thc` | `https://services.upaid.pl/thc` |
| User, Card Data, 3DS and X-Pay Notifications API | Full | `https://sp-api.secure-verestro.dev` | `https://sp-api.secure-verestro.com` |

## 1. HOW FULL API INTEGRATION WORKS

Full API documentation is available here:

Swagger Full API Documentation

In the Full API model, the partner creates the end user and card directly through API.

The process starts with creating a user using:

`POST /secure/users/`

The partner must provide the required user details, including email, phone number, first name, last name, date of birth, and nationality.

After the user is created, Sparados returns `userId`.

The partner then creates an approval for this user and provides the received `userId` in the approval request body. In Full API Integration, after the approval is created, the card is generated automatically and the approval moves directly to `DELIVERED` status.

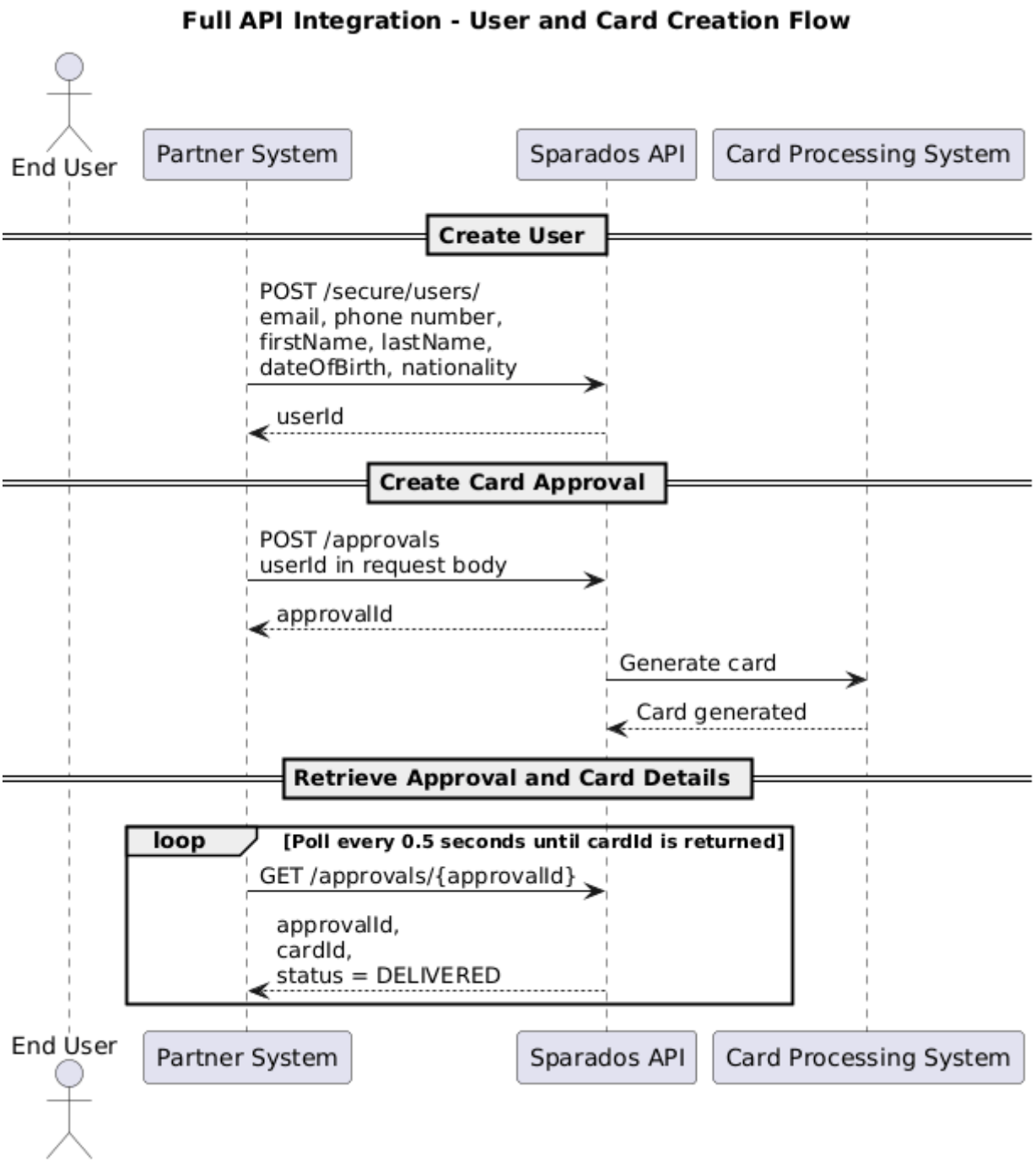
After creating the approval, the partner should poll the approval details endpoint to retrieve the generated `cardId`.

The recommended approach is to call:

`GET /secure/approvals/{approvalId}` every `0.5 second` until `cardId` is returned and the approval status is `DELIVERED`.

Once `cardId` is returned, the partner should store both `approvalId` and `cardId`, as these values are required for later card management and card data retrieval.

The partner should store both `approvalId` and `cardId`, as these values are required for later card management and card data retrieval.




## 2. VALUES USED IN SPARADOS CARDS API

**Swagger documentation is in the end of document**

This section describes common value formats used across Sparados Cards API.
These rules apply when creating approvals, defining card limits, setting validity periods, and managing card-related operations.

#### Date format

Sparados APIs use the ISO 8601 date and time format.

The expected format is:

```
YYYY-MM-DDTHH:mm:ss.SSSZ
```

Where:

- `T` separates the date from the time
- `Z` represents Zulu time / UTC time

Example:

```
2023-05-22T10:00:01.953Z
```

### Minor values

All numeric amount values used in Sparados Cards API are provided in minor units.
For example, to assign a card with a limit of `100.50 EUR`, send:

```
10050
```

This applies to values such as:

- card budget
- additional limits
- budget increases
- budget decreases

### Currency

Card approvals are created in the currency of the account / balance. The currency does not need to be provided separately when creating an approval, because it is inherited from the selected account / balance.
Each account can have only one currency. Transactions may still be performed in other currencies, depending on the card configuration and transaction rules.

## 3. USER MANAGEMENT

In Full API Integration, the partner manages end users directly through API.

The user must be created before an approval can be assigned directly to this user.

### User API Methods

| Method | Endpoint | Description |
|---|---|---|
| `POST` | `/secure/users/` | Creates a new end user. |

| Method | Endpoint | Description |
| --- | --- | --- |
| `GET` | `/secure/users/{userId}` | Retrieves details of a specific user. |
| `PATCH` | `/secure/users/{userId}` | Updates selected user details. |
| `DELETE` | `/secure/users/{userId}` | Deletes or deactivates the user, depending on the agreed configuration. |

### Create User

`POST /secure/users/`

Required fields:

| Field | Type | Description |
| --- | --- | --- |
| `email` | `string` | End user’s email address. |
| `phoneNumber` | `string` | End user’s phone number. |
| `firstName` | `string` | End user’s first name. |
| `lastName` | `string` | End user’s last name. |
| `dateOfBirth` | `string` | End user’s date of birth. Recommended format: `YYYY-MM-DD`. |
| `nationality` | `string` | End user’s nationality. Recommended ISO country code, for example `PL`. |

Example request:

```
{
  "email": "john.smith@example.com",
  "phoneNumber": "48123456789",
  "firstName": "John",
  "lastName": "Smith",
  "dateOfBirth": "1990-01-01",
  "nationality": "PL"
}
```

Example response:

```
{
  "id": 123456
}
```

The returned `userId` should be stored by the partner and used when creating an approval assigned directly to this user.

## 4. CREATING CARD APPROVALS

To create a card for an existing user, use the Approval API.

`POST /secure/approvals`

In Full API Integration, the partner does not provide user contact details in the approval request. The user must be created first through the User API, and the returned `userId` must be provided in the approval request body as `user.id`.

A card approval defines which user should receive the card, what budget can be used, how long the card should be valid, and what transaction limits should apply. After the approval is created and assigned to an existing user, the approval is first created with status `ACCEPTED`. Because the user already exists, the card is generated automatically and the approval then moves directly to `DELIVERED` status.

The API returns `approvalId`. The partner should store this value, as it is required for retrieving card details and managing the approval later.

**Warning**

The `balance.id` used in the Approval API is not the same balance identifier as the `balanceId` used in:

`GET /secure/balances/{balanceId}`

For Approval API methods, including:

`POST /secure/approvals`

you must use the **Card Alias ID** assigned to the account. This value can be copied from the Corporate Panel.

You can find it in the Corporate Panel:

1. Go to **Accounts**.
2. Select the relevant account.
3. Click **Copy Card Alias**.
4. Use the copied value as `balance.id` in the Approval API request.

The `balanceId` required for:

`GET /secure/balances/{balanceId}`

is a different identifier. This value is provided by Sparados each time a new payment account is created.

### Request Body Fields

Required fields are marked with `*`.

| Field | Type | Description |
|---|---|---|
| `balance.id`* | `string` | Balance ID / Card Alias ID assigned to your corporation. This value can be copied from the Corporate Panel. |
| `user.id`* | `integer` | ID of the existing user created through `POST /secure/users/`. |
| `rules.budgetMinor`* | `integer` | Total amount in minor units that the user can spend using the card. |
| `rules.validityPeriod.startDate`* | `string` | Start date and time of the card validity period in ISO 8601 format. |
| `rules.validityPeriod.endDate`* | `string` | End date and time of the card validity period in ISO 8601 format. |
| `rules.validityPeriod.timezone`* | `string` | Timezone used for the validity period, for example `Europe/Warsaw`. |
| `rules.purpose.ecommerce`* | `boolean` | Defines whether the card can be used for e-commerce transactions. Currently, this should be set to `true`. |
| `rules.purpose.allowChangeRequest` | `boolean` | Defines whether the user can request changes to card limits or validity period. |
| `rules.limits` | `array` | List of additional card limits. |
| `card.description` | `string / null` | Description displayed on the card visual. |
| `card.visual.id` | `string` | ID of the card visual. If not provided, the default card visual may be used. |
| `tags` | `array` | List of tag IDs or tag values assigned to the approval. |

### Additional limits

It is possible to set additional limits on the card. These limits are optional. If they are not provided, the card issuing service will apply default values.

There are four main types of additional limits:

**General limit** — periodic limit for all types of transactions. This value should be greater than the other additional limits because it affects them. For example, if the ATM limit is set to `100 EUR` and

the general limit is set to `50 EUR`, the end user will be able to withdraw only `50 EUR` from the ATM.

**Online payment limit** — periodic limit for e-commerce payments.

**ATM limit** — periodic limit for ATM withdrawals.

**Foreign transaction limit** — periodic limit for transactions in a currency different from the account currency. The limit is provided in the account / balance currency and is recalculated using exchange rates and commission.

Possible time units for all additional limits are:

`daily`

`weekly`

`monthly`

Possible values for additional limits:

`0` — disabled. The end user will not be able to perform this type of transaction.

Value in minor units — limited to the requested value.

`null` — unlimited. The end user will be able to perform this type of transaction without a specific additional limit.

**Difference between approval and reapproval**

An approval is created by the corporation in the process of assigning a card to a user.

In Full API Integration, the approval is assigned directly to an existing user by providing `user.id` in the request body.

A reapproval can be created when editing an approval that is already in `DELIVERED` status or when a change request flow is enabled.

## 5. APPROVAL STATUSES AND AVAILABLE ACTIONS

This section describes approval statuses and actions available in Full API Integration.

In Full API Integration, the approval is assigned directly to an existing user by providing `user.id` in the approval request body.

After the approval is created, it is initially created with status `ACCEPTED`. Because the user already exists, the card is generated automatically and the approval then moves directly to `DELIVERED` status.

### Approval Statuses

| Status | Description |
| --- | --- |
| `ACCEPTED` | Approval has been created and assigned to an existing user. This is usually a temporary status before the card is generated. |
| `DELIVERED` | Card has been generated and assigned to the user. This is the active status in which transactions can be made with the card. |
| `FINISHED` | Approval with status `DELIVERED` exceeded the end date of assignment. |
| `REAPPROVED` | Original approval status after a reapproval has been accepted and replaced the previous approval. |
| `LOCKED` | This is a card status, not an approval status. It is shown in the lifecycle because a card can be locked only when the approval is in `DELIVERED` status. |

### Unassigning a card from a user

To remove the card from the user, use:

`PUT /secure/approvals/{id}/unassign`

### Increasing or decreasing available budget

To quickly increase the available budget on a card, use:

`PUT /secure/approvals/{id}/increase_budget`

Example: to add `10 EUR`, send:

```
{
  "additionalBudgetMinor": 1000
}
```

To decrease the available budget, use the same endpoint with a negative value.

Example: to subtract `10 EUR`, send:

```
{
  "additionalBudgetMinor": -1000
}
```

The value must be provided in minor units. For example, `10 EUR = 1000` minor units.

## 6. CHECKING ACCOUNT BALANCE

To check the balance of the payment account connected with issued cards, use:

`GET /secure/balances/{balanceId}`

This method returns the current balance of the account connected with cards.

The `balanceId` required for this method is provided by Sparados after the payment account is created for the partner.

**Important**

The `balanceId` used in this endpoint is different from the `balance.id` used in the Approval API.

For balance checking, use:

`GET /secure/balances/{balanceId}`

For card approval creation, use the **Card Alias ID** copied from the Corporate Panel:

`POST /secure/approvals`

Do not use the Approval API `balance.id` / Card Alias ID in the balance endpoint, and do not use the balance endpoint `balanceId` in Approval API requests.

# 7. CARD DATA RETRIEVAL

To retrieve encrypted card details, use:

`GET /secure/cards/{id}`

This endpoint returns sensitive card data, including PAN and CVV, in encrypted form.

The `{id}` parameter is the `cardId` received from the approval details after the card has been generated.

**Encryption**

Card details are always returned as a JWE token.

To decrypt the response, the partner must provide an RSA public key with each request.

The card data is encrypted using the provided public key. Only the holder of the matching private key can decrypt the response.

**Requirements**

| Requirement | Description |
| --- | --- |

| | |
|---|---|
| RSA key pair | Minimum key size: `2048-bit`. |
| Public key format | Base64-encoded SPKI PEM. |
| Request header | Public key must be sent in the `Public-Key` header. |
| Response format | API returns encrypted payload as a JWE compact token. |
| Encryption algorithm | `RSA-OAEP-256 + A256GCM`. |

### Request Header

| Header | Value |
|---|---|
| `Public-Key` | Base64-encoded RSA public key in SPKI PEM format. |

### Response Format

```
{
  "payload": "<JWE compact token>"
}
```

### Decrypted Payload Example

After decrypting the JWE token, the payload contains card details:

```
{
  "id": 123,
  "type": "VIRTUAL",
  "cardNo": "5414599451242598",
  "cvv": "111",
  "exp": "2031-05-31",
  "issuerCardId": "1111111111111111",
  "dcCorporationId": "7f846a2f-92b4-4094-a776-35cf3753ef51",
  "balanceId": "67716ba1-081b-4919-8aae-9fdbee10be23"
}
```

### Recommended Security Flow

If card data should be visible only to the end user, the key pair should be generated on the frontend side.

Recommended flow:

1. The frontend generates a temporary RSA key pair.
2. The frontend keeps the private key locally, for example in the user session.
3. The frontend sends only the public key to the partner backend.

4. The partner backend forwards the public key to Sparados API in the `Public-Key` header.
5. Sparados returns encrypted card data as a JWE token.
6. The encrypted payload is forwarded back to the frontend.
7. The frontend decrypts the payload using the private key.
8. PAN and CVV are displayed only to the end user.

**Security Note**

Never send the private key to Sparados API or to the partner backend if the decrypted card data should be accessible only to the end user. Only the public key should be sent in the request header.

Accessing, processing, storing, or displaying PAN / CVV may require PCI DSS compliance.

## 8. SPARADOS TRANSACTION API

Sparados Transaction History API allows partners to retrieve transaction history and receive transaction notifications.

The API consists of two parts:

| Type | Description |
|---|---|
| Inbound API | API methods called by the partner to retrieve transaction data from Sparados. |
| Outbound API | Webhook notifications sent by Sparados to the partner system when transaction events occur. |

Transaction API documentation is available here:

`https://developer.verestro.com/books/transaction-history-api/page/technical-documentation-thc-external-api`

**Authentication**

**Inbound Authentication**

Inbound API calls require Mutual TLS authentication.

The same client certificate can be used for all mTLS-secured APIs exposed by Sparados.

If the certificate has not been configured yet, follow the section:

Connecting Server-to-Server

**Outbound Authentication**

For outbound webhook calls, Sparados presents its own server certificate during the TLS handshake.

The certificate is signed by Verestro CA. The partner system should be configured to trust certificates signed by the relevant Verestro CA.

Each environment has its own CA certificate, so the partner must use the correct root CA certificate for the selected environment.

**Backward Compatibility**

Future API changes will be backward compatible. The following changes may be introduced without breaking compatibility:

1. adding a new endpoint
2. adding a new optional request parameter
3. adding a new optional response field
4. adding a new enum value
5. relaxing validation rules for an existing parameter, for example making it optional
6. The partner system should ignore unknown fields and unknown enum values received in API responses.

**Outbound Retry Strategy**

If an outbound webhook call fails because of a timeout, connection issue, or HTTP `5xx` response, Sparados will retry the call automatically.

| Retry stage | Description |
|---|---|
| Initial retries | 3 retries with up to 5 seconds between each attempt. |
| Extended retries | If the call still fails, Sparados performs 3 additional retries after 15 minutes, 30 minutes, and 2 hours. |
| Final result | If all retries fail, the webhook call is dismissed. |

The partner endpoint should return a successful HTTP response after receiving and processing the notification.

# 9. OTP AND WALLET NOTIFICATION WEBHOOKS

In Full API Integration, Sparados can send OTP and wallet-related notifications to endpoints provided by the partner.

The partner must provide Sparados with the full webhook URLs that should be used for notification delivery.

Requests are signed with Verestro’s CA certificate.

The partner endpoint should return:

`204 No Content`

to confirm that the notification was received successfully.

### 3DS OTP Notification

This webhook is triggered when a 3DS OTP is generated for a card in your corporation.

### Endpoint

`POST {3ds_otp_notification_url}`

The full endpoint URL is provided by the partner during integration setup.

### Request Body

| Field | Type | Description |
|---|---|---|
| `balanceId` | `string` | Balance ID related to the card. |
| `currency` | `string` | Transaction currency. |
| `merchantName` | `string` | Merchant name for the transaction. |
| `otp` | `string` | 3DS one-time password generated for the transaction. |
| `cardId` | `integer` | Card ID. |
| `userId` | `integer` | User ID assigned to the card. |
| `amountMinor` | `integer` | Transaction amount in minor units. |
| `cardLast4Digits` | `string` | Last 4 digits of the card number. |

### Example Request Body

```
{
  "balanceId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "currency": "EUR",
  "merchantName": "Example Merchant",
  "otp": "123456",
  "cardId": 123,
  "userId": 456,
  "amountMinor": 1000,
  "cardLast4Digits": "2598"
}
```

### Expected Response

`204 No Content`

### Wallet Notifications

This webhook is triggered when a wallet notification event occurs for a card in your corporation.

It is used for Apple Pay / Google Pay related events, including activation code delivery and token status updates.

### Endpoint

`POST {wallet_notifications_url}`

The full endpoint URL is provided by the partner during integration setup.

### Message Types

| `messageType` | Description |
|---|---|
| `ACTIVATION_CODE_DELIVERY` | Wallet activation code should be delivered to the end user. |
| `TOKEN_ACTIVATED` | Wallet token has been activated. |
| `CARD_REMOVED_BY_CUSTOMER` | Card has been removed from the wallet by the customer. |
| `CARD_REMOVED_BY_ISSUER` | Card has been removed from the wallet by the issuer. |

### Request Body

| Field | Type | Description |
|---|---|---|
| `cardLast4Digits` | `string` | Last 4 digits of the card number. |
| `tokenRequestorType` | `string` | Wallet provider / token requestor type. |
| `messageType` | `string` | Wallet notification type. |
| `activationMethodType` | `string` | Activation method type. Present only for `ACTIVATION_CODE_DELIVERY`. |
| `activationMethodValue` | `string` | Activation method value. Present only for `ACTIVATION_CODE_DELIVERY`. |
| `activationCode` | `string` | Wallet activation code. Present only for `ACTIVATION_CODE_DELIVERY`. |
| `formFactor` | `string` | Device or wallet form factor. |
| `cardId` | `integer` | Card ID. |

### Example Request Body

```
{
  "cardLast4Digits": "2598",
  "tokenRequestorType": "APPLE_PAY",
  "messageType": "ACTIVATION_CODE_DELIVERY",
  "activationMethodType": "SMS",
  "activationMethodValue": "+48123456789",
  "activationCode": "123456",
  "formFactor": "PHONE",
  "cardId": 123
}
```

### Expected Response

`204 No Content`

### Important

The fields `activationCode`, `activationMethodType`, and `activationMethodValue` are present only when `messageType` is:

`ACTIVATION_CODE_DELIVERY`

## 10. API SWAGGER DOCUMENTATION

Detailed API documentation is available here:

| API | Documentation link |
|---|---|
| Full API Swagger | `https://sp-api.verestro.dev/docs?urls.primaryName=External` |
| Cards API Swagger | `https://sparados-bc-api.upaidtest.pl/api-secure.yaml` |
| Transaction API Documentation | `https://developer.verestro.com/books/transaction-history-api/page/technical-documentation-thc-external-api` |

Use the Swagger documentation as the source of detailed endpoint schemas, request parameters, response formats, and available enum values.

---

Revision #8
Created 25 May 2026 13:55:33 by Michał Stachera
Updated 26 May 2026 12:21:33 by Michał Stachera