SPRIBE Games Integration API (1.9.0)

Game Launch Process

To launch a game, online casino should generate launch link with appropriate parameters and open it in browser.

- Production
- Demo

Parameter	Type	Description
game	string [game enum]	Identifies specific game. (Game Identificator)
user	string	ID of a player on the operators side
token	string	The one time token is a unique string generated by the ope
currency	string [currency enum]	Game currency (currency code)
operator	string	operator key name
lang optional	string [lang enum]	Game interface language
return_url optional	string [pattern url]	URL for return from game
account_history_url optional	string [pattern url]	URL for the players bet history in the game menu and the re
irc_duration optional	integer [seconds]	Period to stop the game and show the reality check window
irc_elapsed optional	integer [seconds]	The elapsed time from the first period of the reality check.

URL Structure

https://{launch-url}/{game}?user={user}&token={token}&lang={lang}&currency={currency}&operator={operator}&return_url={return_url}

Specifications

Games

Name	Provider key	Game identificator
Aviator	spribe_aviator	aviator
Dice	spribe_crypto	dice
Goal	spribe_crypto	goal
Plinko	spribe_crypto	plinko

Name	Provider key	Game identificator
Mines	spribe_crypto	mines
Hi Lo	spribe_crypto	hi-lo
Keno	spribe_crypto	keno
Mini Roulette	spribe_crypto	mini-roulette
Hotline	spribe_crypto	hotline
Balloon	spribe_crypto	balloon
Keno 80	spribe_keno	multikeno
Trader	spribe_trader	trader
Crystal Fall	spribe_slots	crystal-fall
Neo Vegas	spribe_slots	neo-vegas
Gates of Egypt	spribe_slots	gates-of-egypt

Authentication

This is a method that consists in requesting an authentication for a player who is trying to launch the game

- Path: **/auth**
- Request Method: **POST**
- Content-Type: **application/json; charset=utf-8**
- Headers: **X-Spribe-Client-ID, X-Spribe-Client-TS, X-Spribe-Client-Signature**

Request

- Parameters
- Sample

Parameter	Type	Description
user_token	string	Token generated by operator during game l
session_token	string	Token generated by provider for current ga
platform	string [platform enum]	Enum: mobiledesktop
currency	string [currency enum]	Game currency. (currency code)

Response

- Parameters
- Success sample
- Error sample

Parameter	Type	Description
code	integer [enum]	Response code
message	string	Response any logical message
data	object	only needed when code 200
user_id	string	ID of player on operator side
username	string	Name of player
balance	long	Current balance of player in specified curre
currency	string [currency enum]	Currency of player

Important!

Each request sent to the API requires validation using specific security headers. For details on implementing this, refer to Securing API Requests.

Amount/Balance example

Amount/Balance fiat is represented in units(1$ = 1000 unit) and always is an integer value. For example: with 5.32 USD and the value will be 5.32 * 10^3 = 5320

Amount/Balance crypto is represented in units(1BTC = 100000000 unit) and always is an integer value. For example: with 0.0532 BTC and the value will be 0.0532 * 10^8 = 5320000

Possible codes

- **200** - Success
- **401** - User token is invalid
- **403** - User token is expired
- **413** - Invalid Client-Signature
- **500** - Internal error

If code is **200** - operator should always return parameter data with information; HTTP status code should always be **200**

Player information

This method retrieves player information

- Path: **/info**
- Request Method: **POST**
- Content-Type: **application/json; charset=utf-8**

- Headers: **X-Spribe-Client-ID, X-Spribe-Client-TS, X-Spribe-Client-Signature**

Request

- Parameters
- Sample

Parameter	Type	Description
user_id	string	Player ID on operator side
session_token	string	Token generated by provider for current ga
currency	string [currency enum]	Game currency. (currency code)

Response

- Parameters
- Success sample
- Error sample

Parameter	Type	Description
code	integer [enum]	Response code
message	string	Response any logical message
data	object	only needed when code 200
user_id	string	ID of player on operator side
username	string	Name of player
balance	long	Current balance of player in specified curre
currency	string [currency enum]	Currency of player

Important!

Each request sent to the API requires validation using specific security headers. For details on implementing this, refer to Securing API Requests.

Possible codes

- **200** - Success
- **401** - User token is invalid
- **403** - User token is expired
- **413** - Invalid Client-Signature
- **500** - Internal error

If code is **200** - operator should always return parameter data with information; HTTP status code should always be **200**

Withdraw

This method withdraws money from player balance. Method sends one transaction and returns transaction ID, player balance and status of transaction.

- Path: **/withdraw**
- Request Method: **POST**
- Content-Type: **application/json; charset=utf-8**
- Headers: **X-Spribe-Client-ID, X-Spribe-Client-TS, X-Spribe-Client-Signature**

Request

- Parameters
- Sample

Parameter	Type	Description
user_id	string	Player ID on operator side
currency	string [currency enum]	Bet currency (currency code)
amount	long	Amount that should be transferred from pl
provider	string	Game provider
provider_tx_id	string	The transaction ID assigned by provider
game	string [game enum]	Identifies specific game. (game identificato
action	string [action enum]	Enum: betrain
action_id	string	ID of action in game, which depends on gar
session_token	string	Game session token
platform	string [platform enum]	Enum: mobiledesktop

Response

- Parameters
- Success sample
- Duplication sample
- Error sample

Parameter	Type	Description
code	integer [enum]	Response code

Parameter	Type	Description
message	string	Response any logical message
data	object	only needed when code 200 or 409
user_id	string	ID of a player on the operators side
operator_tx_id	string	Unique transaction ID on side of operator
provider	string	Game provider
provider_tx_id	string	The transaction ID received by provider
old_balance	long	Balance before deposit
new_balance	long	Balance after deposit
currency	string [currency enum]	Transaction currency

Important!

Each request sent to the API requires validation using specific security headers. For details on implementing this, refer to Securing API Requests.

If request is timed out, game sends same transaction, with same provider_tx_id again. if transaction is already processed on the side of the operator, operator identifies duplication and adds in answer duplication error code with already processed information of transaction. The Transaction should be unique with provider_tx_id.

If operator processed transaction successfully, but after this, provider can't process it in the game, provider will rollback transaction with rollback method.

Amount/Balance example

Amount/Balance fiat is represented in units(1$ = 1000 unit) and always is an integer value. For example: with 5.32 USD and the value will be 5.32 * 10^3 = 5320

Amount/Balance crypto is represented in units(1BTC = 100000000 unit) and always is an integer value. For example: with 0.0532 BTC and the value will be 0.0532 * 10^8 = 5320000

Possible codes

- **200** - Success
- **401** - User token is invalid
- **402** - Insufficient fund
- **403** - User token is expired
- **405** - Internal error with no retry
- **409** - Duplicate transaction
- **412** - For stop the game and show the reality check window. In the window will shown the message from the response. (read more)

- **413** - Invalid Client-Signature
- **500** - Internal error

If code is **200** - operator should always return parameter data with information; HTTP status code should always be **200**

Deposit

This method makes a deposit into players account. The method sends transaction and returns transaction ID, player balance and status of transaction.

- Path: **/deposit**
- Request Method: **POST**
- Content-Type: **application/json; charset=utf-8**
- Headers: **X-Spribe-Client-ID, X-Spribe-Client-TS, X-Spribe-Client-Signature**

Request

- Parameters
- Sample

Parameter	Type	Description
user_id	string	Player ID
currency	string [currency enum]	Bet currency (currency code)
amount	long	Amount that should be transferred to playe
provider	string	Game provider
provider_tx_id	string	The transaction ID assigned by provider
game	string [game enum]	Identifies specific game. (game identificato
action	string	Type of action: betrainfreebetrainfreebetpr
action_id	string	ID of action in game, which depends on gar
session_token	string	Game session token
platform	string [platform enum]	Enum: mobiledesktop
withdraw_provider_tx_id optional	string	Provider id of withdrawn transaction, to wh

Response

- Parameters
- Success sample
- Duplication sample

- Error sample

Parameter	Type	Description
code	integer [enum]	Response code
message	string	Response any logical message
data	object	only needed when code 200 or 409
user_id	string	ID of a player on the operators side
operator_tx_id	string	Unique transaction ID on side of operator
provider	string	Game provider
provider_tx_id	string	The transaction ID received by provider
old_balance	long	Balance before deposit
new_balance	long	Balance after deposit
currency	string [currency enum]	Transaction currency

Important!

Each request sent to the API requires validation using specific security headers. For details on implementing this, refer to Securing API Requests.

If request gets on timeout, or an internal error, game sends same transaction, with same provider_tx_id again. If transaction is already processed on the side of the operator, operator identifies duplication and adds in answer duplication error code with processed information of transaction. The Transaction should be unique with provider_tx_id.

Amount/Balance example

Amount/Balance fiat is represented in units(1$ = 1000 unit) and always is an integer value. For example: with 5.32 USD and the value will be 5.32 * 10^3 = 5320

Amount/Balance crypto is represented in units(1BTC = 100000000 unit) and always is an integer value. For example: with 0.0532 BTC and the value will be 0.0532 * 10^8 = 5320000

Possible codes

- **200** - Success
- **401** - User token is invalid
- **403** - User token is expired
- **409** - Duplicate transaction
- **413** - Invalid Client-Signature
- **500** - Internal error

If code is **200** - operator should always return parameter data with information; HTTP status code should always be **200**

Rollback

This method rollbacks transaction with provider transaction ID.

- Path: **/rollback**
- Request Method: **POST**
- Content-Type: **application/json; charset=utf-8**
- Headers: **X-Spribe-Client-ID, X-Spribe-Client-TS, X-Spribe-Client-Signature**

Request

- Parameters
- Sample

Parameter	Type	Description
user_id	string	Player ID
amount	long	Amount of money that should be rollbacke
provider	string	Game provider
rollback_provider_tx_id	string	Provider transaction ID which should be rol
provider_tx_id	string	The transaction ID assigned by provider
game	string [game enum]	Identifies specific game. (game identificato
session_token	string	Game session token
action	string [action enum]	Enum: betrain
action_id	string	ID of action in game, which depends on gar

Response

- Parameters
- Success sample
- Duplication sample
- Error sample

Parameter	Type	Description
code	integer [enum]	Response code
message	string	Response any logical message

Parameter	Type	Description
data	object	only needed when code 200 or 409
user_id	string	ID of a player on the operators side
currency	string [currency enum]	Transaction currency
operator_tx_id	string	Transaction ID on side of operator
provider	string	Game provider
provider_tx_id	string	The transaction ID assigned by provider
old_balance	long	Balance before deposit
new_balance	long	Balance after deposit

Important!

Each request sent to the API requires validation using specific security headers. For details on implementing this, refer to Securing API Requests.

If request gets on timeout, or an internal error, game sends same transaction, with same provider_tx_id again. If transaction is already processed on the side of the operator, operator identifies duplication and adds in answer duplication error code with processed information of transaction. The Transaction should be unique with provider_tx_id.

Amount/Balance example

Amount/Balance fiat is represented in units(1$ = 1000 unit) and always is an integer value. For example: with 5.32 USD and the value will be 5.32 * 10^3 = 5320

Amount/Balance crypto is represented in units(1BTC = 100000000 unit) and always is an integer value. For example: with 0.0532 BTC and the value will be 0.0532 * 10^8 = 5320000

Possible codes

- **200** - Success
- **401** - User token is invalid
- **403** - User token is expired
- **408** - Transaction does not found
- **409** - Duplicate transaction
- **413** - Invalid Client-Signature
- **500** - Internal error

If code is **200** - operator should always return parameter data with information; HTTP status code should always be **200**

Reality check

A Reality Check gives to the players the option of setting a time frequency at which you will receive an notification about the length of time you have spent in game.

To stop the game and show the reality check notification, the API should on **withdraw** method answer with the error code 412. In the window will be shown the message from the response.e.g { "code": 412, "message": "You have been playing for 60 minutes" }. In message should be information about the length of the time in game. Besides message, in the window following buttons will be presented: **stop** , **continue** & **account history**. When player presses the button **stop** or **continue** the game calls **playerNotificationCallback** method. When player presses the button **account history**, the game opens the link of the game history of players, which was presented in launch URL;

Important!

Currently reality check is available only for AVIATOR

Securing API Requests

To maintain secure and verified communication between systems, all requests made between our API and operators must adhere to the following security requirements. The specifics below outline how requests should be validated, whether initiated by the API or the operator.

Security Requirements for API Requests

For requests originating from our API to operators, operators should validate each incoming request by checking the following headers. Similarly, operators should add these headers to requests they initiate to our API.

Required Headers for Request Validation

Each secured request must contain the following headers:

1. **X-Spribe-Client-ID:** A unique identifier for the operator, provided by us.
2. **X-Spribe-Client-TS:** A timestamp in UTC, representing seconds since the Unix Epoch.
3. **X-Spribe-Client-Signature:** A hashed signature that verifies the integrity of the request parameters.

Generating the X-Spribe-Client-Signature Header

The X-Spribe-Client-Signature ensures request authenticity by following these steps to create a signature:

1. **Concatenate the following values:**
    - **Timestamp** (X-Spribe-Client-TS): Timestamp at the time of the request.
    - **Request URI:** URL path with ordered and encoded query parameters (without the domain).
    - **Request Body:** For POST and PUT requests, include the request body as a string. Omit this for GET and DELETE requests.

2. **Hash the concatenated string:** Use the SHA256 HMAC algorithm, with the operator's Client Secret as the key, to hash the concatenated string.

Validation Process

Upon receiving a request, the server (operator's or ours) should validate the X-Spribe-Client-Signature by:

1. Retrieving the Client Secret based on X-Spribe-Client-ID.
2. Repeating the signature creation process using the headers and parameters.
3. Comparing the generated signature to the received X-Spribe-Client-Signature header. A match indicates the request is verified.

Example Implementation for Incoming API Requests

Below is an example illustrating how to implement secure request handling on the operator's side or ours:

```
@PostMapping(path = "/secure-request")
public ResponseEntity<Void> secureRequest(@RequestBody final byte[] body,
                                          @RequestHeader(name = "X-Spribe-Client-ID") final String clientId,
                                          @RequestHeader(name = "X-Spribe-Client-TS") final long time,
                                          @RequestHeader(name = "X-Spribe-Client-Signature") final String signature,
                                          final HttpServletRequest request) {
    // Validate headers and timestamp expiration
    final var clientSecret = getClientSecretFromRepository(clientId); // Retrieve Client Secret
    final var queryString = request.getQueryString();
    final var path = request.getRequestURI() + (queryString == null || "".equals(queryString) ? "" : "?" + request.getQueryString());
    final var calculatedSignature = createSignature(time, path, body, clientSecret);

    if (!signature.equalsIgnoreCase(calculatedSignature)) {
        throw new SecurityException("Signature mismatch.");
    }
}

public static String createSignature(final long time, final String path, final byte[] body, final String clientSecret) {
    try {
```

```
        Mac sha256Hmac = Mac.getInstance("HmacSHA256");
        sha256Hmac.init(new SecretKeySpec(clientSecret.getBytes(StandardCharsets.UTF_8),
"HmacSHA256"));
        sha256Hmac.update((time + path).getBytes(StandardCharsets.UTF_8));


        byte[] bytes = body == null ? sha256Hmac.doFinal() : sha256Hmac.doFinal(body);
        return HexFormat.of().formatHex(bytes);
    } catch (Exception e) {
        throw new SecurityException("Error creating signature.", e);
    }
}
```

This structure clarifies that both the operator and the API system are responsible for including and validating security headers on their outgoing and incoming requests, ensuring consistent security standards across all communications.