Event Checkout & Payment API

Author: Josh S. Sakweli, Backend Lead Team **Last Updated**: 2026-05-23 **Version**: v2.0

Base URL: `https://api.nextgate.co.tz/api/v1`

Short Description: The NextGate Checkout API manages the complete ticket purchasing lifecycle on the NextGate event platform. It supports two distinct checkout flows: online checkout for registered attendees and at-door ticket sales for event organizers and scanner devices. This API should be used by frontend clients, mobile applications, and authorized scanner hardware integrations.

Hints:

- All monetary values are in **TZS (Tanzanian Shilling)** unless otherwise stated
- Checkout sessions expire after **15 minutes** for online checkout and **1 hour** for at-door sales — always check `expiresAt` before processing payment
- Bearer token authentication is required for all endpoints
- **Wallet balance is validated at session creation time for PAID tickets** — if insufficient, the session is never created and a structured `422` response is returned with `shortfall` and `recommendedTopUp` so the frontend can direct the user to top up immediately
- For FREE tickets, payment is auto-processed immediately upon session creation — no separate payment step is needed
- For DONATION tickets, maximum 1 ticket per order and cannot be purchased for other attendees
- Ticket holds are applied immediately on session creation; cancelling the session releases the hold
- Scanner devices must have the `SELL_TICKETS` permission assigned and a valid `deviceFingerprint` registered before calling the scanner sale endpoint

---

Standard Response Format

All API responses follow a consistent structure using our Globe Response Builder pattern:

Success Response Structure

```
{
  "success": true,
  "httpStatus": "OK",
```

```
  "message": "Operation completed successfully",
  "action_time": "2025-09-23T10:30:45",
  "data": {}
}
```

Error Response Structure

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Error description",
  "action_time": "2025-09-23T10:30:45",
  "data": "Error description"
}
```

Insufficient Balance Error Structure (422)

Returned when wallet balance is insufficient **at session creation time** for a PAID ticket. No session is created. The `data` field carries rich balance details so the frontend can guide the user directly to top up.

```
{
  "success": false,
  "httpStatus": "UNPROCESSABLE_ENTITY",
  "message": "Insufficient wallet balance to complete checkout",
  "action_time": "2025-09-23T10:30:45",
  "data": {
    "walletBalance": 50000.00,
    "sessionTotal": 150000.00,
    "shortfall": 100000.00,
    "hasSufficientBalance": false,
    "recommendedTopUp": 100000.00,
    "pspMinimum": 500.00,
    "currency": "TZS"
  }
}
```

Field	Description
`walletBalance`	Current wallet balance of the user in TZS

Field	Description
`sessionTotal`	Total amount required for this checkout
`shortfall`	Amount missing (`sessionTotal - walletBalance`)
`hasSufficientBalance`	Always `false` when this error is returned
`recommendedTopUp`	Suggested top-up amount — at minimum equals `shortfall`, rounded up to `pspMinimum` if needed
`pspMinimum`	Minimum top-up amount accepted by the payment provider
`currency`	Always `TZS`

Standard Response Fields

Field	Type	Description
`success`	boolean	Always `true` for successful operations, `false` for errors
`httpStatus`	string	HTTP status name (OK, BAD_REQUEST, NOT_FOUND, etc.)
`message`	string	Human-readable message describing the operation result
`action_time`	string	ISO 8601 timestamp of when the response was generated
`data`	object/string	Response payload for success, error details for failures

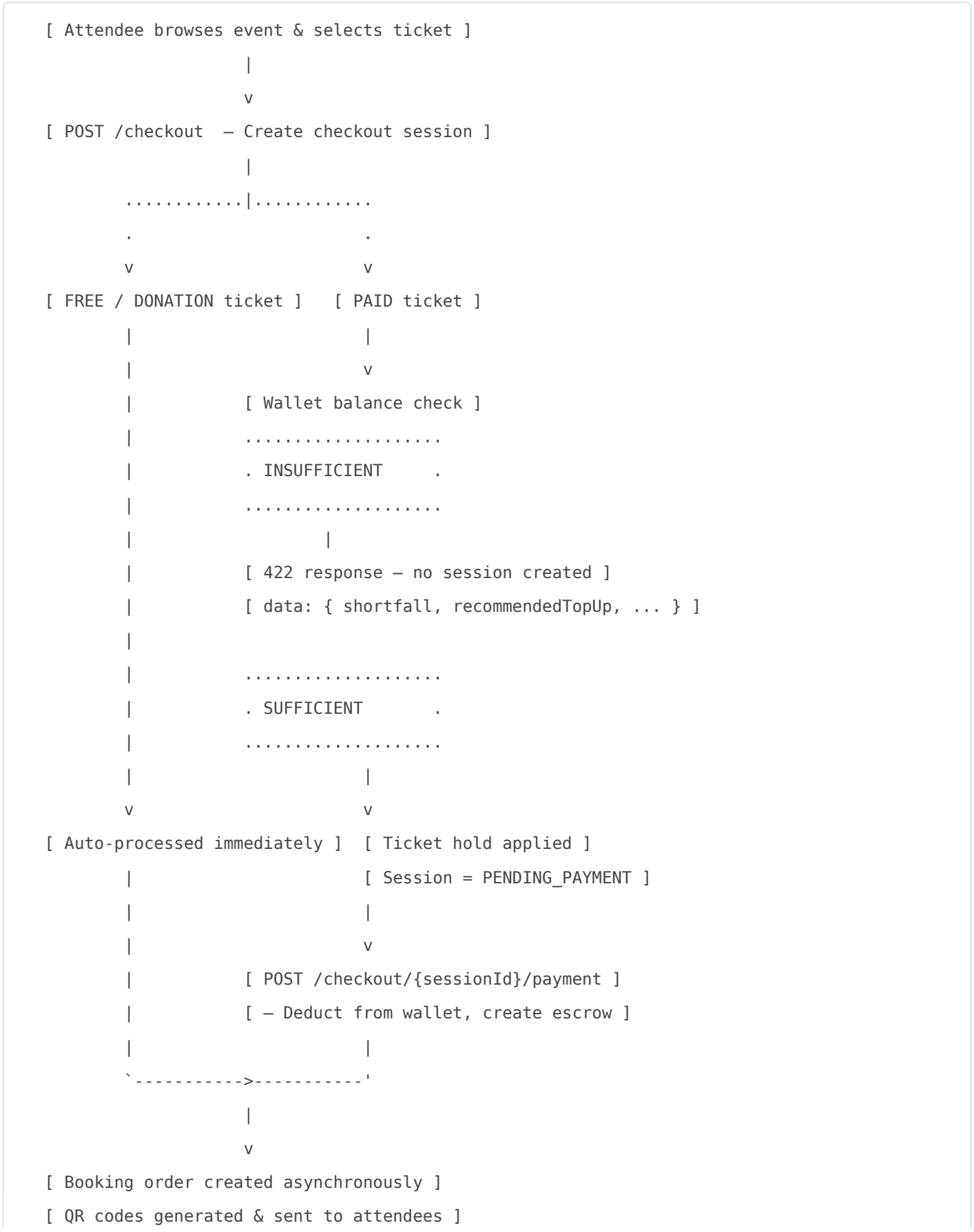
---

HTTP Method Badge Standards

- **GET** — Green: Safe, read-only operations
- **POST** — Blue: Create new resources
- **PUT** — Yellow: Update/replace entire resource
- **PATCH** — Orange: Partial updates
- **DELETE** — Red: Remove resources

User Journey Flows

Flow A — Online Checkout (Registered Attendee)



```
                    |
                    v
[ Session status = COMPLETED ]



At any point before payment:
[ POST /checkout/{sessionId}/cancel ]
[ — Releases ticket hold            ]
```

Flow B — At-Door Sale via Scanner Device

```
[ Customer arrives at event venue ]
                    |
                    v
[ Scanner device sends sale request ]
[ POST /checkout/sell-at-door-ticket/scanner ]
                    |
        ............|............
        .                        .
        v                        v
[ Validate scanner ID ]   [ Validate device fingerprint ]
[ & permissions       ]   [ & SELL_TICKETS permission   ]
        .                        .
        `..........v............'
                    |
                    v
[ Validate ticket type belongs to event ]
[ Check sales channel = AT_DOOR or BOTH ]
                    |
                    v
[ Cash payment processed (no wallet deduction) ]
[ Booking order created immediately            ]
                    |
                    v
[ QR codes returned in response ]
[ immediateCheckIn = true → ticket marked as checked-in ]
```

Flow C — At-Door Sale via Organizer

```
[ Organizer is authenticated & accesses event ]
                    |
                    v
[ POST /checkout/{eventId}/organizer ]
[ — Organizer sells ticket at their counter ]
                    |
                    v
[ System verifies organizer owns the event ]
                    |
                    v
[ Validate ticket type & attendee count ]
                    |
                    v
[ Cash payment recorded (NEUTRAL transaction) ]
[ Booking order created                       ]
                    |
                    v
[ QR codes returned in response ]
[ immediateCheckIn flag respected ]
```

Endpoints

---

1. Create Checkout Session

Purpose: Creates a new online checkout session for a registered attendee purchasing event tickets, holding the requested quantity and initializing the payment intent.

Endpoint: `POST` `/api/v1/e-events/checkout`

Access Level: □□ Protected (Requires valid Bearer token — authenticated attendee)

Authentication: Bearer Token

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token of the authenticated attendee
`Content-Type`	string	Yes	Must be `application/json`

Request JSON Sample:

```
{
  "eventId": "b3f1a2c4-1234-5678-abcd-000000000001",
  "ticketTypeId": "d4e5f6a7-1234-5678-abcd-000000000002",
  "ticketsForMe": 2,
  "donationAmount": null,
  "otherAttendees": [
    {
      "name": "Jane Doe",
      "email": "jane.doe@example.com",
      "phone": "+255712345678",
      "quantity": 1
    }
  ],
  "sendTicketsToAttendees": true,
  "paymentMethodId": null
}
```

Request Body Parameters:

Parameter	Type	Required	Description	Validation
`eventId`	UUID	Yes	The ID of the event being booked	Must be a valid published event that has not yet started
`ticketTypeId`	UUID	Yes	The ID of the ticket type being purchased	Must belong to the specified event and be active and on sale
`ticketsForMe`	integer	Yes	Number of tickets for the buyer themselves. Use `0` if the buyer is not attending	Min: 0
`donationAmount`	decimal	No	Donation amount in TZS. Only applicable for `DONATION` type tickets	Only used when ticket pricing type is `DONATION`

Parameter	Type	Required	Description	Validation
`otherAttendees`	array	No	List of other attendees to purchase tickets for	Each attendee must have valid name, email, and Tanzanian phone number
`otherAttendees[].name`	string	Yes (if array provided)	Full name of the attendee	Min: 2, Max: 100 characters
`otherAttendees[].email`	string	Yes (if array provided)	Email address of the attendee	Valid email format; no duplicate emails in the array
`otherAttendees[].phone`	string	Yes (if array provided)	Phone number of the attendee	Must match Tanzania format: `+255[67]XXXXXXXX`
`otherAttendees[].quantity`	integer	Yes (if array provided)	Number of tickets for this attendee	Min: 1
`sendTicketsToAttendees`	boolean	No	If `true`, QR tickets are sent to each attendee's email. If `false`, all QR codes are sent to the buyer only	Default: `true`
`paymentMethodId`	UUID	No	ID of a saved payment method. Defaults to wallet if not provided	Optional

Business Rules:

- Total quantity = `ticketsForMe` + sum of all `otherAttendees[].quantity` — must be at least 1
- `DONATION` tickets: maximum 1 per order, cannot be bought for other attendees, online only
- `AT_DOOR_ONLY` tickets cannot be purchased through this endpoint
- Wallet balance is validated upfront for `PAID` tickets
- If the event has a required questionnaire set to `BEFORE_CHECKOUT`, it must be submitted before calling this endpoint
- `FREE` tickets are auto-processed immediately — the response will already show `PAYMENT_COMPLETED`

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "CREATED",
  "message": "Checkout session created successfully",
  "action_time": "2025-09-23T10:30:45",
```

```
"data": {
  "sessionId": "a1b2c3d4-0000-0000-0000-000000000001",
  "status": "PENDING_PAYMENT",
  "customerId": "f1e2d3c4-0000-0000-0000-000000000010",
  "customerUserName": "john_doe",
  "eventId": "b3f1a2c4-1234-5678-abcd-000000000001",
  "eventTitle": "Kilimanjaro Jazz Night 2025",
  "ticketDetails": {
    "ticketTypeId": "d4e5f6a7-1234-5678-abcd-000000000002",
    "ticketTypeName": "VIP",
    "unitPrice": 50000.00,
    "ticketsForBuyer": 2,
    "otherAttendees": [
      {
        "name": "Jane Doe",
        "email": "jane.doe@example.com",
        "phone": "+255712345678",
        "quantity": 1
      }
    ],
    "sendTicketsToAttendees": true,
    "totalQuantity": 3,
    "subtotal": 150000.00
  },
  "pricing": {
    "subtotal": 150000.00,
    "total": 150000.00
  },
  "paymentIntent": {
    "provider": "WALLET",
    "clientSecret": null,
    "paymentMethods": ["WALLET"],
    "status": "PENDING"
  },
  "ticketsHeld": true,
  "ticketHoldExpiresAt": "2025-09-23T10:45:45",
  "expiresAt": "2025-09-23T10:45:45",
  "createdAt": "2025-09-23T10:30:45",
  "updatedAt": "2025-09-23T10:30:45",
  "completedAt": null,
```

```
    "createdBookingOrderId": null,
    "isExpired": false,
    "canRetryPayment": false
  }
}
```

Success Response Fields:

Field	Description
`sessionId`	Unique identifier for this checkout session. Use it for all subsequent actions
`status`	Current session status. Values: `PENDING_PAYMENT`, `PAYMENT_COMPLETED`, `COMPLETED`, `PAYMENT_FAILED`, `CANCELLED`, `EXPIRED`
`customerId`	Account ID of the buyer
`customerUserName`	Username of the buyer
`eventId`	ID of the event being booked
`eventTitle`	Human-readable event title
`ticketDetails.ticketTypeId`	ID of the chosen ticket type
`ticketDetails.ticketTypeName`	Name of the ticket type (e.g., VIP, Regular)
`ticketDetails.unitPrice`	Price per single ticket in TZS
`ticketDetails.ticketsForBuyer`	Number of tickets allocated to the buyer
`ticketDetails.otherAttendees`	List of other attendees and their ticket quantities
`ticketDetails.sendTicketsToAttendees`	Whether QR codes will be emailed to each attendee
`ticketDetails.totalQuantity`	Total tickets across buyer and all attendees
`ticketDetails.subtotal`	Total price before any discounts (TZS)
`pricing.subtotal`	Subtotal amount in TZS
`pricing.total`	Final payable amount in TZS
`paymentIntent.provider`	Payment provider (e.g., `WALLET`)
`paymentIntent.paymentMethods`	Available payment methods for this session
`paymentIntent.status`	Payment intent status (`PENDING`, `COMPLETED`)
`ticketsHeld`	Whether the tickets are currently being held in reserve
`ticketHoldExpiresAt`	Timestamp when the ticket hold expires
`expiresAt`	Timestamp when the entire session expires
`createdBookingOrderId`	Populated after payment is completed — the resulting booking order ID

Field	Description
`isExpired`	Computed flag indicating whether the session has passed its expiry time
`canRetryPayment`	Whether the session allows another payment attempt (true if status is `PAYMENT_FAILED` and attempts < 5)

Error Response JSON Sample:

```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Please complete the event questionnaire before purchasing tickets",
  "action_time": "2025-09-23T10:30:45",
  "data": "Please complete the event questionnaire before purchasing tickets"
}
```

Possible Errors:

HTTP Status	Scenario
`400 BAD_REQUEST`	Event is not published, event has already started, ticket not on sale, AT_DOOR_ONLY ticket purchased online, DONATION rules violated, questionnaire not submitted
`401 UNAUTHORIZED`	Missing, expired, or invalid Bearer token
`404 NOT_FOUND`	Event or ticket type not found
`422 UNPROCESSABLE_ENTITY`	Insufficient wallet balance (returns rich `data` with `shortfall` and `recommendedTopUp`); or validation failed on request fields
`500 INTERNAL_SERVER_ERROR`	Unexpected server error

---

2. Get Checkout Session

Purpose: Retrieves the current state of an existing checkout session belonging to the authenticated user.

Endpoint: `GET` `/api/v1/e-events/checkout/{sessionId}`

Access Level: □□ Protected (Authenticated attendee — only the session owner can retrieve it)

Authentication: Bearer Token

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token of the authenticated attendee

Path Parameters:

Parameter	Type	Required	Description	Validation
`sessionId`	UUID	Yes	The ID of the checkout session to retrieve	Must be a valid UUID belonging to the authenticated user

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Checkout session retrieved successfully",
  "action_time": "2025-09-23T10:35:00",
  "data": {
    "sessionId": "a1b2c3d4-0000-0000-0000-000000000001",
    "status": "PENDING_PAYMENT",
    "customerId": "f1e2d3c4-0000-0000-0000-000000000010",
    "customerUserName": "john_doe",
    "eventId": "b3f1a2c4-1234-5678-abcd-000000000001",
    "eventTitle": "Kilimanjaro Jazz Night 2025",
    "ticketDetails": {
      "ticketTypeId": "d4e5f6a7-1234-5678-abcd-000000000002",
      "ticketTypeName": "VIP",
      "unitPrice": 50000.00,
      "ticketsForBuyer": 2,
      "otherAttendees": [
        {
          "name": "Jane Doe",
          "email": "jane.doe@example.com",
          "phone": "+255712345678",
          "quantity": 1
        }
      ],
      "sendTicketsToAttendees": true,
```

```
      "totalQuantity": 3,
      "subtotal": 150000.00
    },
    "pricing": {
      "subtotal": 150000.00,
      "total": 150000.00
    },
    "paymentIntent": {
      "provider": "WALLET",
      "clientSecret": null,
      "paymentMethods": ["WALLET"],
      "status": "PENDING"
    },
    "paymentAttempts": [],
    "ticketsHeld": true,
    "ticketHoldExpiresAt": "2025-09-23T10:45:45",
    "expiresAt": "2025-09-23T10:45:45",
    "createdAt": "2025-09-23T10:30:45",
    "updatedAt": "2025-09-23T10:30:45",
    "completedAt": null,
    "createdBookingOrderId": null,
    "isExpired": false,
    "canRetryPayment": false
  }
}
```


Success Response Fields:

Field	Description
`sessionId`	Unique session identifier
`status`	Current session status
`paymentAttempts`	List of all payment attempts made on this session, including failures
`paymentAttempts[].attemptNumber`	Sequential attempt number (1-indexed)
`paymentAttempts[].paymentMethod`	Payment method used for this attempt
`paymentAttempts[].status`	Result of the attempt: `SUCCESS` or `FAILED`
`paymentAttempts[].errorMessage`	Failure reason if the attempt failed
`paymentAttempts[].attemptedAt`	Timestamp of the attempt
`paymentAttempts[].transactionId`	External or internal transaction reference

Field	Description
All other fields	Same as Create Checkout Session response

Possible Errors:

HTTP Status	Scenario
`401 UNAUTHORIZED`	Missing, expired, or invalid Bearer token
`404 NOT_FOUND`	Session not found or does not belong to the authenticated user
`500 INTERNAL_SERVER_ERROR`	Unexpected server error

---

3. Process Payment

Purpose: Initiates wallet payment for a pending checkout session, creating an escrow account and triggering asynchronous booking order creation upon success.

Endpoint: **POST** `/api/v1/e-events/checkout/{sessionId}/payment`

Access Level: □□ Protected (Session owner only)

Authentication: Bearer Token

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token of the authenticated attendee

Path Parameters:

Parameter	Type	Required	Description	Validation
`sessionId`	UUID	Yes	The ID of the checkout session to pay for	Must be a valid UUID; session must be in `PENDING_PAYMENT` status and not expired

Business Rules:

- Session must be in `PENDING_PAYMENT` status
- Session must not be expired
- Cannot call this on FREE tickets (auto-processed on creation)

- Wallet must have sufficient balance at time of payment call
- Maximum 5 payment attempts per session; after that `canRetryPayment` becomes `false`
- On success, escrow is created, session moves to `PAYMENT_COMPLETED`, and a booking order is created asynchronously

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Payment completed successfully. Your booking is being processed.",
  "action_time": "2025-09-23T10:38:00",
  "data": {
    "success": true,
    "status": "SUCCESS",
    "message": "Payment completed successfully. Your booking is being processed.",
    "checkoutSessionId": "a1b2c3d4-0000-0000-0000-000000000001",
    "escrowId": "e9f8a7b6-0000-0000-0000-000000000020",
    "escrowNumber": "ESC-2025-000001",
    "orderId": null,
    "orderNumber": null,
    "paymentMethod": "WALLET",
    "amountPaid": 150000.00,
    "platformFee": 7500.00,
    "sellerAmount": 142500.00,
    "currency": "TZS"
  }
}
```


Success Response Fields:

Field	Description
`status`	Payment result status: `SUCCESS`, `FAILED`, or `PENDING`
`checkoutSessionId`	The checkout session this payment belongs to
`escrowId`	ID of the escrow account holding the funds
`escrowNumber`	Human-readable escrow reference number (format: ESC-YYYY-NNNNNN)
`orderId`	Booking order ID — may be `null` immediately after payment as booking is created asynchronously
`orderNumber`	Human-readable order reference — `null` until order is created

Field	Description
`paymentMethod`	Payment method used: `WALLET`
`amountPaid`	Total amount deducted from buyer's wallet in TZS
`platformFee`	Platform fee amount (5% of total) in TZS
`sellerAmount`	Amount that will be released to the event organizer in TZS
`currency`	Currency code, always `TZS`

Possible Errors:

HTTP Status	Scenario
`400 BAD_REQUEST`	Session is not in PENDING_PAYMENT status, or session is expired
`401 UNAUTHORIZED`	Missing, expired, or invalid Bearer token
`404 NOT_FOUND`	Session not found or does not belong to the authenticated user
`500 INTERNAL_SERVER_ERROR`	Unexpected payment processing or booking creation error

---

4. Cancel Checkout Session

Purpose: Cancels an active checkout session and releases any held tickets back to available inventory.

Endpoint: **POST** `/api/v1/e-events/checkout/{sessionId}/cancel`

Access Level: □□ Protected (Session owner only)

Authentication: Bearer Token

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token of the authenticated attendee

Path Parameters:

Parameter	Type	Required	Description	Validation

`sessionId`	UUID	Yes	The ID of the checkout session to cancel	Must be a valid UUID belonging to the authenticated user

Business Rules:

- Cannot cancel a session that is in `COMPLETED` status
- Cannot cancel a session that is in `PAYMENT_COMPLETED` status (payment already processed)
- Cancelling releases the ticket hold immediately
- Cancellation does not trigger a refund — refunds are handled separately through the escrow system

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "OK",
  "message": "Checkout session cancelled successfully",
  "action_time": "2025-09-23T10:40:00",
  "data": null
}
```

Possible Errors:

HTTP Status	Scenario
`400 BAD_REQUEST`	Session is already completed or payment has been completed
`401 UNAUTHORIZED`	Missing, expired, or invalid Bearer token
`404 NOT_FOUND`	Session not found or does not belong to the authenticated user
`500 INTERNAL_SERVER_ERROR`	Unexpected server error

---

5. Scanner — Sell Ticket at Door

Purpose: Allows an authorized scanner device to sell tickets at the venue entrance, processing a cash payment and optionally checking in the attendee immediately.

Endpoint: **POST** `/api/v1/e-events/checkout/sell-at-door-ticket/scanner`

Access Level: □□ Protected (Scanner device authentication via `scannerId` + `deviceFingerprint`)

Authentication: Bearer Token (of scanner's linked account) + Scanner credentials in body

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token linked to the scanner's registered account
`Content-Type`	string	Yes	Must be `application/json`

Request JSON Sample:

```
{
  "scannerId": "SCN-2025-001",
  "deviceFingerprint": "a3f1b2c4d5e6f7890abc1234def56789",
  "ticketTypeId": "d4e5f6a7-1234-5678-abcd-000000000002",
  "quantity": 2,
  "attendees": [
    {
      "fullName": "John Mbeki",
      "email": "john.mbeki@example.com",
      "phoneNumber": "+255789123456"
    },
    {
      "fullName": "Amina Hassan",
      "email": "amina.hassan@example.com",
      "phoneNumber": "+255754321987"
    }
  ],
  "immediateCheckIn": true
}
```

Request Body Parameters:

Parameter	Type	Required	Description	Validation
`scannerId`	string	Yes	Unique identifier of the registered scanner device	Must match a registered, active scanner with `SELL_TICKETS` permission
`deviceFingerprint`	string	Yes	Hardware fingerprint of the scanner device	Must match the fingerprint registered for this scanner

Parameter	Type	Required	Description	Validation
`ticketTypeId`	UUID	Yes	ID of the ticket type to sell	Must belong to the event this scanner is assigned to; must not be `ONLINE_ONLY`; must be on sale
`quantity`	integer	Yes	Total number of tickets to sell	Min: 1; must equal the number of attendees in the `attendees` array
`attendees`	array	Yes	List of attendee details, one entry per ticket	Min: 1 entry; count must match `quantity`
`attendees[].fullName`	string	No	Full name of the attendee	Optional — if blank, a generated name like `ATTENDEE-XXXX` is assigned
`attendees[].email`	string	No	Email address of the attendee	Valid email format if provided
`attendees[].phoneNumber`	string	No	Phone number of the attendee	Optional
`immediateCheckIn`	boolean	Yes	If `true`, the ticket is marked as checked-in immediately upon sale	Required

Business Rules:

- The scanner must be active, not expired, and have the `SELL_TICKETS` permission
- The `deviceFingerprint` must exactly match the registered fingerprint for this scanner
- The number of attendees must equal `quantity` — a 1-to-1 mapping is enforced
- Payment method is always `CASH` — no wallet deduction occurs
- The ticket type must not be `ONLINE_ONLY`
- `immediateCheckIn = true` automatically marks each generated ticket as checked-in

Success Response JSON Sample:


```
{
  "success": true,
  "httpStatus": "CREATED",
  "message": "Tickets sold successfully at door",
  "action_time": "2025-09-23T18:00:00",
  "data": {
    "bookingId": "c9d8e7f6-0000-0000-0000-000000000030",
    "bookingReference": "BK-2025-000042",
```

```
    "eventId": "b3f1a2c4-1234-5678-abcd-000000000001",
    "eventName": "Kilimanjaro Jazz Night 2025",
    "tickets": [
      {
        "ticketInstanceId": "aa11bb22-0000-0000-0000-000000000050",
        "ticketSeries": "VIP-0042-A",
        "ticketTypeName": "VIP",
        "attendeeName": "John Mbeki",
        "attendeeEmail": "john.mbeki@example.com",
        "checkedIn": true,
        "checkInTime": "2025-09-23T18:00:05Z",
        "qrCode": "eyJhbGciOiJIUzI1NiJ9..."
      },
      {
        "ticketInstanceId": "cc33dd44-0000-0000-0000-000000000051",
        "ticketSeries": "VIP-0042-B",
        "ticketTypeName": "VIP",
        "attendeeName": "Amina Hassan",
        "attendeeEmail": "amina.hassan@example.com",
        "checkedIn": true,
        "checkInTime": "2025-09-23T18:00:05Z",
        "qrCode": "eyJhbGciOiJIUzI1NiJ9..."
      }
    ],
    "totalAmount": 100000.00,
    "currency": "TZS",
    "paymentMethod": "CASH",
    "soldBy": "Gate-A Scanner",
    "soldAt": "Main Entrance",
    "saleTime": "2025-09-23T18:00:05Z"
  }
}
```


Success Response Fields:

Field	Description
`bookingId`	UUID of the created booking order
`bookingReference`	Human-readable booking reference number
`eventId`	ID of the event

Field	Description
`eventName`	Name of the event
`tickets`	Array of issued ticket instances — one per attendee
`tickets[].ticketInstanceId`	Unique ID of this specific ticket instance
`tickets[].ticketSeries`	Ticket serial number (e.g., VIP-0042-A)
`tickets[].ticketTypeName`	The type of the sold ticket
`tickets[].attendeeName`	Name of the attendee this ticket is assigned to
`tickets[].attendeeEmail`	Email of the attendee
`tickets[].checkedIn`	Whether the attendee has been checked in
`tickets[].checkInTime`	Timestamp of check-in if `immediateCheckIn` was `true`
`tickets[].qrCode`	JWT-encoded QR code string for this ticket
`totalAmount`	Total cash amount collected in TZS
`currency`	Always `TZS`
`paymentMethod`	Always `CASH` for at-door sales
`soldBy`	Name of the scanner that processed the sale
`soldAt`	Location label of the scanner (e.g., "Main Entrance")
`saleTime`	ISO 8601 timestamp of when the sale occurred

Possible Errors:

HTTP Status	Scenario
`400 BAD_REQUEST`	Attendee count does not match quantity, ticket is ONLINE_ONLY, ticket not on sale
`401 UNAUTHORIZED`	Missing or invalid Bearer token
`403 FORBIDDEN`	Scanner does not have SELL_TICKETS permission, device fingerprint mismatch, scanner is inactive or expired
`404 NOT_FOUND`	Scanner ID not found, ticket type not found
`422 UNPROCESSABLE_ENTITY`	Validation errors on request fields
`500 INTERNAL_SERVER_ERROR`	Payment processing or booking creation failure

---

6. Organizer — Sell Ticket at Door

Purpose: Allows the authenticated event organizer to sell tickets directly at their event counter, processing a cash payment and optionally checking in the attendee immediately.

Endpoint: `POST` `/api/v1/e-events/checkout/sell-at-door-ticket/{eventId}/organizer`

Access Level: □□ Protected (Must be the organizer of the specified event)

Authentication: Bearer Token

Request Headers:

Header	Type	Required	Description
`Authorization`	string	Yes	Bearer token of the authenticated event organizer
`Content-Type`	string	Yes	Must be `application/json`

Path Parameters:

Parameter	Type	Required	Description	Validation
`eventId`	UUID	Yes	The ID of the event to sell tickets for	Must be an existing, non-deleted event; authenticated user must be the organizer

Request JSON Sample:

```
{
  "ticketTypeId": "d4e5f6a7-1234-5678-abcd-000000000002",
  "quantity": 2,
  "attendees": [
    {
      "fullName": "Peter Salim",
      "email": "peter.salim@example.com",
      "phoneNumber": "+255711223344"
    },
    {
      "fullName": "Grace Mwangi",
      "email": "grace.mwangi@example.com",
      "phoneNumber": null
    }
  ],
  "immediateCheckIn": false,
  "location": "VIP Gate"
}
```

Request Body Parameters:

Parameter	Type	Required	Description	Validation
`ticketTypeId`	UUID	Yes	ID of the ticket type to sell	Must belong to the event in the path; must not be `ONLINE_ONLY`; must be on sale
`quantity`	integer	Yes	Total number of tickets to sell	Min: 1; must equal the number of attendees in the `attendees` array
`attendees`	array	Yes	List of attendee details — one entry per ticket	Min: 1 entry; count must match `quantity`
`attendees[].fullName`	string	No	Full name of the attendee	Optional — auto-generated if blank
`attendees[].email`	string	No	Email of the attendee	Valid email format if provided
`attendees[].phoneNumber`	string	No	Phone number of the attendee	Optional
`immediateCheckIn`	boolean	Yes	Whether to mark attendees as checked-in immediately	Required
`location`	string	No	Description of the sale point, e.g., "VIP Gate", "Main Counter"	Max: 200 characters; defaults to `"Organizer Counter"` if not provided

Business Rules:

- Only the event organizer (the user who created the event) can call this endpoint
- Number of entries in `attendees` must equal `quantity`
- Payment is always `CASH` — no wallet or ledger deduction
- Ticket type must not be `ONLINE_ONLY`
- `immediateCheckIn = true` marks each ticket as checked-in at time of sale

Success Response JSON Sample:

```
{
  "success": true,
  "httpStatus": "CREATED",
  "message": "Tickets sold successfully at door",
  "action_time": "2025-09-23T17:30:00",
```

```
  "data": {
    "bookingId": "d7e6f5a4-0000-0000-0000-000000000035",
    "bookingReference": "BK-2025-000043",
    "eventId": "b3f1a2c4-1234-5678-abcd-000000000001",
    "eventName": "Kilimanjaro Jazz Night 2025",
    "tickets": [
      {
        "ticketInstanceId": "ee55ff66-0000-0000-0000-000000000060",
        "ticketSeries": "VIP-0043-A",
        "ticketTypeName": "VIP",
        "attendeeName": "Peter Salim",
        "attendeeEmail": "peter.salim@example.com",
        "checkedIn": false,
        "checkInTime": null,
        "qrCode": "eyJhbGciOiJIUzI1NiJ9..."
      },
      {
        "ticketInstanceId": "gg77hh88-0000-0000-0000-000000000061",
        "ticketSeries": "VIP-0043-B",
        "ticketTypeName": "VIP",
        "attendeeName": "Grace Mwangi",
        "attendeeEmail": "grace.mwangi@example.com",
        "checkedIn": false,
        "checkInTime": null,
        "qrCode": "eyJhbGciOiJIUzI1NiJ9..."
      }
    ],
    "totalAmount": 100000.00,
    "currency": "TZS",
    "paymentMethod": "CASH",
    "soldBy": "organizer_username",
    "soldAt": "VIP Gate",
    "saleTime": "2025-09-23T17:30:05Z"
  }
}
```


Success Response Fields:

Field	Description
`bookingId`	UUID of the created booking order

Field	Description
`bookingReference`	Human-readable booking reference number
`eventId`	ID of the event
`eventName`	Name of the event
`tickets`	Array of issued ticket instances — one per attendee
`tickets[].ticketInstanceId`	Unique ID of this specific ticket instance
`tickets[].ticketSeries`	Ticket serial number
`tickets[].ticketTypeName`	The type of ticket sold
`tickets[].attendeeName`	Assigned attendee name
`tickets[].attendeeEmail`	Attendee email
`tickets[].checkedIn`	Whether immediately checked in
`tickets[].checkInTime`	Check-in timestamp, `null` if not checked in
`tickets[].qrCode`	JWT-encoded QR code string for this ticket
`totalAmount`	Total cash amount in TZS
`currency`	Always `TZS`
`paymentMethod`	Always `CASH`
`soldBy`	Username of the organizer who made the sale
`soldAt`	Location label provided in the request
`saleTime`	ISO 8601 timestamp of the sale

Possible Errors:

HTTP Status	Scenario
`400 BAD_REQUEST`	Attendee count does not match quantity, ticket is ONLINE_ONLY, ticket not on sale
`401 UNAUTHORIZED`	Missing or invalid Bearer token
`403 FORBIDDEN`	Authenticated user is not the organizer of the specified event
`404 NOT_FOUND`	Event not found, ticket type not found
`422 UNPROCESSABLE_ENTITY`	Validation errors on request fields
`500 INTERNAL_SERVER_ERROR`	Payment processing or booking creation failure

Standard Error Response Examples

Bad Request — General (400):


```
{
  "success": false,
  "httpStatus": "BAD_REQUEST",
  "message": "Ticket is not currently on sale",
  "action_time": "2025-09-23T10:30:45",
  "data": "Ticket is not currently on sale"
}
```


Unauthorized — Token Issues (401):


```
{
  "success": false,
  "httpStatus": "UNAUTHORIZED",
  "message": "Token has expired",
  "action_time": "2025-09-23T10:30:45",
  "data": "Token has expired"
}
```


Forbidden — Access Denied (403):


```
{
  "success": false,
  "httpStatus": "FORBIDDEN",
  "message": "Only the event organizer can sell tickets at door",
  "action_time": "2025-09-23T10:30:45",
  "data": "Only the event organizer can sell tickets at door"
}
```


Not Found (404):


```
{
  "success": false,
  "httpStatus": "NOT_FOUND",
  "message": "Event not found",
  "action_time": "2025-09-23T10:30:45",
  "data": "Event not found"
}
```


Validation Error (422):

```
{
  "success": false,
  "httpStatus": "UNPROCESSABLE_ENTITY",
  "message": "Validation failed",
  "action_time": "2025-09-23T10:30:45",
  "data": {
    "ticketTypeId": "must not be null",
    "quantity": "must be greater than or equal to 1",
    "immediateCheckIn": "must not be null"
  }
}
```

Standard Error Types Reference

Application-Level Exceptions (400–499)

- `400 BAD_REQUEST`: General invalid request, business rule violations, or item already exists
- `401 UNAUTHORIZED`: Authentication issues (missing, invalid, expired, or malformed token)
- `403 FORBIDDEN`: Access denied, scanner permission issues, organizer mismatch
- `404 NOT_FOUND`: Event, ticket type, session, or scanner not found
- `422 UNPROCESSABLE_ENTITY`: Bean validation errors with per-field details
- `429 TOO_MANY_REQUESTS`: Rate limit exceeded

Server-Level Exceptions (500+)

- `500 INTERNAL_SERVER_ERROR`: Unexpected server errors, payment orchestration failures

---

Ticket Pricing Types — Detailed Behaviour

This section explains exactly how the system handles each pricing type end-to-end: from checkout creation through payment, booking order creation, and ticket serial assignment. Understanding this is critical for integrating correctly with the checkout API.

FREE Tickets

What they are: Tickets with a price of `0 TZS`. No money changes hands.

Checkout flow:

```
[ POST /checkout — session created ]
            |
            v
[ System detects price = 0 TZS ]
            |
            v
[ Payment auto-processed immediately ]
[ No wallet deduction               ]
[ No escrow created                 ]
            |
            v
[ PaymentCompletedEvent published (escrow = null) ]
            |
            v
[ Booking order created asynchronously ]
[ Ticket serials assigned              ]
            |
            v
[ Session status = COMPLETED ]
[ Response returned to caller ]
```

Key rules:

- The caller does **not** need to call `POST /{sessionId}/payment` — this is skipped entirely
- The session is returned from the create endpoint already in `PAYMENT_COMPLETED` status (may shift to `COMPLETED` once the booking is written)
- No escrow record exists for this transaction; `escrowId` will be `null` in all responses
- A `NEUTRAL` transaction history entry is recorded for audit purposes
- Ticket holds are still applied on creation and released naturally upon booking completion
- FREE tickets can be `ONLINE_ONLY` or `BOTH` depending on the sales channel configuration — `AT_DOOR_ONLY` FREE tickets go through the scanner/organizer at-door flows instead

---

PAID Tickets

What they are: Tickets with a price greater than `0 TZS`. Wallet payment is required.

Checkout flow:

```
[ PaymentCompletedEvent published (escrow != null) ]
            |
            v
[ Booking order created asynchronously ]
[ Ticket serials assigned              ]
[ QR codes generated                   ]
            |
            v
[ Session status = COMPLETED ]
[ Escrow status = HELD       ]
[ (Released to organizer on event completion) ]
```

Key rules:

- Wallet balance is checked **at session creation** — if insufficient, a `422` is returned with rich balance data (`shortfall`, `recommendedTopUp`) and no session is created. A second safety-net check runs at actual payment time in case the balance changed between the two calls
- Maximum **5 payment attempts** per session; after that `canRetryPayment = false` and a new session must be created
- If a payment attempt fails, the session moves to `PAYMENT_FAILED` but the ticket hold remains active until the session expires
- Escrow holds funds in a separate ledger account — the organizer does **not** receive the money until the platform releases it after the event
- `orderId` in the payment response may be `null` immediately after payment since booking creation is asynchronous — poll `GET /checkout/{sessionId}` and check `createdBookingOrderId` to confirm

---

DONATION Tickets

What they are: Tickets where the attendee voluntarily chooses the amount they pay. A minimum may or may not be set by the organizer.

Checkout flow:

```
[ POST /checkout — session created ]
[ donationAmount provided in body  ]
            |
            v
[ System validates DONATION rules  ]
.....................................
```

```
. totalQuantity must be exactly 1  .
. otherAttendees must be empty     .
. salesChannel must be ONLINE_ONLY .
.....................................
            |
            v
[ Treated as PAID internally                    ]
[ donationAmount used as the ticket price       ]
[ Wallet deducted for the donation amount       ]
[ Escrow created for the donation amount        ]
            |
            v
[ Booking order created asynchronously ]
[ Ticket serial assigned               ]
[ QR code generated                    ]
            |
            v
[ Session status = COMPLETED ]
```

Key rules:

- **Strictly 1 ticket per order** — the system rejects any request with `ticketsForMe > 1` or any `otherAttendees`
- **Online-only** — DONATION tickets cannot be sold at the door through any channel
- The `donationAmount` in the request body is the amount that will be charged; the system uses it as the effective unit price
- Despite being a donation, the standard 5% platform fee still applies and an escrow account is created
- If `donationAmount` is `null` or `0`, the system may treat it as a FREE ticket depending on the ticket's configured minimum — confirm with the organizer's ticket setup

---

Ticket Serials — How They Are Assigned

Every ticket instance issued by the system receives a unique **ticket serial** (also called `ticketSeries` in the response). This serial is the human-readable identifier printed on physical tickets, displayed in QR codes, and used for manual verification at the door.

Serial Format

```
[TICKET_TYPE_CODE]-[BOOKING_NUMBER]-[POSITION_LETTER]

Examples:
  VIP-0042-A      ← First VIP ticket in booking #42
  VIP-0042-B      ← Second VIP ticket in booking #42
  REG-0199-A      ← First Regular ticket in booking #199
  GENERAL-0001-A  ← First General Admission ticket in booking #1
```

How Serials Are Generated

```
[ Booking order created ]
            |
            v
[ System reads totalQuantity from checkout session ]
            |
            v
[ For each ticket in the order: ]
.....................................
.  ticketSeries = TYPE_CODE         .
.               + "-"               .
.               + BOOKING_NUMBER    .  ← zero-padded (e.g., 0042)
.               + "-"               .
.               + POSITION_LETTER   .  ← A, B, C, D ... per ticket
.....................................
            |
            v
[ Each serial stored on the TicketInstance entity ]
[ JWT-encoded QR token generated per ticket       ]
[ QR token embeds: ticketSeries + ticketInstanceId ]
            |
            v
[ Serials returned in: ]
.  POST /sell-at-door-ticket response (tickets[].ticketSeries) ]
.  Booking order details endpoint (separate booking API)       ]
```

Serial Assignment per Pricing Type

Pricing Type	When Serials Are Assigned	Who Appears in `attendeeName`

`FREE`	Asynchronously after `PaymentCompletedEvent`	Buyer for `ticketsForBuyer` tickets; each named attendee for their tickets
`PAID`	Asynchronously after payment escrow is created	Buyer for `ticketsForBuyer` tickets; each named attendee for their tickets
`DONATION`	Asynchronously after payment (always 1 ticket)	Always the buyer only
At-Door (any type)	Synchronously — returned immediately in the sale response	Each attendee in the `attendees` array; auto-generated name if blank

Attendee-to-Serial Mapping

When a buyer purchases tickets for themselves and other attendees, each serial maps to exactly one person:

```
Buyer purchases:
  ticketsForMe = 2
  otherAttendees = [ { name: "Jane", quantity: 1 } ]
  totalQuantity = 3

Serials assigned:
  VIP-0042-A  →  Buyer (ticket 1 of 2 for buyer)
  VIP-0042-B  →  Buyer (ticket 2 of 2 for buyer)
  VIP-0042-C  →  Jane  (her 1 ticket)
```

QR Code & Serial Relationship

Each ticket's `qrCode` field in the response is a **JWT token** that encodes the ticket serial and instance ID. Scanners decode this JWT at check-in time to verify the ticket. The serial alone is readable by humans; the JWT is what the scanner hardware validates cryptographically.

```
QR Code (JWT) decodes to:
.......................................
.  ticketInstanceId  (UUID)            .
.  ticketSeries      (e.g. VIP-0042-A)  .
.  eventId           (UUID)            .
.  issuedAt          (timestamp)       .
.......................................
            |
            v
[ Scanner validates JWT signature ]
```

```
[ Marks ticket as CHECKED_IN      ]
[ Returns check-in confirmation   ]
```

Quick Reference

Endpoint Summary

#	Method	Path	Description
1	POST	`/api/v1/e-events/checkout`	Create online checkout session
2	GET	`/api/v1/e-events/checkout/{sessionId}`	Get checkout session details
3	POST	`/api/v1/e-events/checkout/{sessionId}/payment`	Process wallet payment
4	POST	`/api/v1/e-events/checkout/{sessionId}/cancel`	Cancel checkout session
5	POST	`/api/v1/e-events/checkout/sell-at-door-ticket/scanner`	Scanner at-door sale
6	POST	`/api/v1/e-events/checkout/sell-at-door-ticket/{eventId}/organizer`	Organizer at-door sale

Session Status Reference

Status	Meaning
`PENDING_PAYMENT`	Session created, awaiting payment
`PAYMENT_PROCESSING`	External payment initiated, awaiting confirmation
`PAYMENT_COMPLETED`	Payment succeeded, booking being created
`PAYMENT_FAILED`	Payment attempt failed (retry may be possible)
`COMPLETED`	Booking fully created and confirmed
`CANCELLED`	Session cancelled by user
`EXPIRED`	Session timed out before payment

Ticket Pricing Type Behaviour

Pricing Type	Payment Required	At-Door Allowed	Notes
`FREE`	No	Depends on sales channel	Auto-processed on session creation
`PAID`	Yes (Wallet)	Yes	Escrow created on payment
`DONATION`	Optional amount	No (Online only)	Max 1 ticket per order; no other attendees

Sales Channel Rules

Sales Channel	Online Checkout	At-Door (Scanner)	At-Door (Organizer)
`ONLINE_ONLY`	✅ Allowed	❌ Blocked	❌ Blocked
`AT_DOOR_ONLY`	❌ Blocked	✅ Allowed	✅ Allowed
`BOTH`	✅ Allowed	✅ Allowed	✅ Allowed

Authentication Reference

- **Bearer Token**: Include `Authorization: Bearer <token>` in all request headers
- All endpoints require authentication
- For scanner endpoints, the Bearer token must belong to the account linked to the scanner device

Data Format Standards

- **Dates**: ISO 8601 format (`2025-09-23T10:30:45`)
- **Currency**: TZS (Tanzanian Shilling) — decimal values with 2 decimal places
- **UUIDs**: Standard UUID v4 format (`xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`)
- **Phone Numbers**: Tanzania format only — `+255[67]XXXXXXXX`

Revision #6
Created 11 December 2025 10:03:03 by Admin Qbit
Updated 23 May 2026 05:43:07 by Admin Qbit