Guia de Integracion

CloudRooms + HyperGuest

Implementacion en Node.js (REST + Webhooks)

Version 1.0 - 2026-02-06

Documento tecnico para equipos de integracion (Back-end).

Control del documento

Campo	Valor
Producto	CloudRooms - Conector HyperGuest
Audiencia	Equipo tecnico (Node.js) / Integraciones
Version	1.0
Fecha	2026-02-06
Estado	Borrador utilizable (lista de endpoints + ejemplos)

Nota: Este manual describe el contrato tecnico (REST) que tu servicio Node.js debe consumir para buscar disponibilidad, cotizar, reservar, consultar, modificar y cancelar. Tambien incluye webhooks y patrones de idempotencia, reintentos y control de errores.

Tabla de contenido

1. Alcance y supuestos

Este documento asume que CloudRooms se integra como **travel provider / demand** hacia el marketplace de HyperGuest y que tu back-end en Node.js expone una API interna para tu front (web/app) y para tus procesos (reconciliacion, post-venta, etc.).

Si tu integracion es del lado **accommodation provider / supply** (PMS/Channel Manager), la estructura es similar pero cambia el sentido de algunos flujos (push ARI).

1.1 Perfil del partner (segun cuestionario)

Pregunta	Respuesta registrada / asumida
Seleccionar nacionalidad del huesped en busqueda	Si (se envia como parametro en /search/availability)
Modelo de negocio (B2B / B2C / Both)	Both (si no se confirma, ajustar en configuracion)
Escenario 'pay at hotel'	No especificado - el manual incluye soporte
Escenario 'paid by agent'	No especificado - el manual incluye soporte
Tipos de pago soportados	Tarjeta del cliente, VCC por reserva, saldo a credito, pago externo (transfere
Monedas soportadas	USD, EUR, GBP, ARS, COP, Otros
Tarifas de paquete dinamico	No especificado - tratar como feature opcional

Si algun campo anterior no coincide con tu configuracion real, solo cambia el **modo de pago** o flags de negocio en tu capa de integracion. Los endpoints base no cambian.

2. Arquitectura recomendada

Se recomienda un servicio dedicado ("connector") en Node.js que concentre autenticacion, rate limiting, reintentos, idempotencia, trazabilidad y adaptacion de modelos entre tu negocio y CloudRooms/HyperGuest.

Componente	Responsabilidad
Front (Web/App)	Busqueda, seleccion, checkout. Llama a tu API interna.
API interna (Node.js)	Orquesta flujos y persiste ordenes/reservas.
Connector CloudRooms	Wrapper de endpoints REST + Webhooks (este manual).
HyperGuest	Marketplace y conectividad con accommodation providers.

HyperGuest indica que soporta API moderna para datos dinamicos (bookable/ARI) en modo PULL y PUSH, y API para datos estaticos (propiedad/habitacion). Esto encaja con el conector descrito aqui.

3. Entornos, URLs base y versionado

Define estas variables en tu servicio (por entorno):

```
CLOUDROOMS_BASE_URL=https://api.cloudrooms.example/v1
CLOUDROOMS_API_KEY=********
CLOUDROOMS_API_SECRET=******** # si se usa firma HMAC
TIMEOUT_MS=15000
```

Convencion: todos los endpoints de este manual se publican bajo **/v1**. Si hay un cambio incompatible se publicara **/v2** y se mantendra compatibilidad por una ventana acordada.

4. Seguridad y autenticacion

Se soportan dos opciones. Recomendada: **API Key + firma HMAC** (mejor control anti-replay). Alternativa: **Bearer token** (OAuth2 o token de integracion).

4.1 API Key + firma HMAC

Headers requeridos:

```
X-CR-API-Key: <api_key>
X-CR-Timestamp: <unix_epoch_ms>
X-CR-Signature: <hex_hmac_sha256>
Content-Type: application/json
Idempotency-Key: <uuid> # solo en POST/PUT con efecto
```

String a firmar (ASCII):

```
<timestamp>\n<HTTP_METHOD>\n<REQUEST_PATH>\n<SHA256_HEX_OF_BODY>
```

Ejemplo de firma (Node.js):

```
import crypto from "crypto";

export function signRequest({secret, timestamp, method, path, body}) {
const bodyHash = crypto.createHash("sha256").update(body || "").digest("hex");
const payload = `${timestamp}\n${method.toUpperCase()}\n${path}\n${bodyHash}`;
return crypto.createHmac("sha256", secret).update(payload).digest("hex");
}
```

4.2 Idempotencia

Para evitar dobles cobros o dobles reservas en reintentos, envia **Idempotency-Key** en creacion de booking, pagos, modificaciones y cancelaciones. El servidor debe devolver la misma respuesta para la misma llave durante la ventana de idempotencia.

5. Convenciones de datos

Codigos recomendados:

Dato	Formato
Pais/Nacionalidad	ISO 3166-1 alpha-2 (ej: CO, US, AR)
Moneda	ISO 4217 (ej: COP, USD, EUR)
Fecha	YYYY-MM-DD
Hora	RFC3339 (ej: 2026-02-06T12:34:56-05:00)

6. API REST - Endpoints que debes usar

Todos los ejemplos usan JSON y HTTPS. Las rutas se expresan relativas a **CLOUDROOMS_BASE_URL**.

6.1 Health y version

```
GET /health
GET /v1/meta/version
```

Usalos para monitoreo y para validar compatibilidad durante despliegues.

6.2 Datos estaticos (contenido)

Estos endpoints se usan para poblar catalogo (propiedades, habitaciones, politicas, fotos, etc.).

```
GET /v1/properties?updatedSince=YYYY-MM-DD
GET /v1/properties/{propertyId}
GET /v1/properties/{propertyId}/rooms
GET /v1/properties/{propertyId}/rateplans
```

6.3 Busqueda y disponibilidad (incluye nacionalidad)

```
POST /v1/search/availability
```

Request (ejemplo):

```
{
"checkIn": "2026-03-10",
"checkOut": "2026-03-12",
"currency": "COP",
"nationality": "CO",
"residencyCountry": "CO",
"properties": [
{
"propertyId": "HG-12345"
}
],
"occupancies": [
{
"adults": 2,
"children": [
7
]
}
],
"filters": {
"mealPlan": [
"AI",
"BB"
],
"refundableOnly": false
},
"clientReference": "VX-123456"
}
```

Response (resumen): retorna una lista de **offers** con precio total, impuestos, politica de cancelacion, forma de pago permitida y un **offerId** para cotizar/reservar.

6.4 Cotizacion (bloqueo logico de precio)

```
POST /v1/quote
```

Usa este endpoint para convertir un offerId en un quoteId con vigencia corta (ej: 5-15 min).

```
{
"offerId": "OFF-abc123",
"currency": "COP",
"clientReference": "VX-123456"
}
```

6.5 Crear reserva

```
POST /v1/bookings
```

Campos clave: quoteId (o offerId), datos del titular, pasajeros, contacto, y **payment**.

Ejemplo (pay at hotel):

```
{
"quoteId": "QTE-7890",
"partnerBookingRef": "VX-123456",
"holder": {
"firstName": "Hector",
"lastName": "Sanchez",
"email": "holder@example.com",
"phone": "+57 3000000000"
},
"guests": [
{
"type": "ADULT",
"firstName": "Hector",
"lastName": "Sanchez",
"nationality": "CO"
}
],
"specialRequests": "Late check-in",
"payment": {
"mode": "PAY_AT_HOTEL",
"guarantee": "CARD_REQUIRED",
"currency": "COP"
}
}
```

Modos de pago soportados segun tu cuestionario:

Modo	Cuando usarlo	Notas
PAY_AT_HOTEL	El cliente paga en el hotel	Normalmente requiere garantia (tarjeta).
CUSTOMER_CARD	Cobro en linea con tarjeta del cliente	Requiere tokenizacion/PCI segun tu flujo.
VCC_PER_BOOKING	Pago por agente con VCC emitida por reserva	Se envia/recibe VCC segun acuerdo.
CREDIT_BALANCE	Se descuenta de un cupo/saldo acordado	Se reconcilia contra settlement.
EXTERNAL_TRANSFER	Transferencia bancaria / pago externo	Se adjuntan instrucciones y evidencia.

6.6 Consultar reserva

```
GET /v1/bookings/{bookingId}
GET /v1/bookings?partnerBookingRef=VX-123456
```

Usalo para post-venta, soporte, y para confirmar el estado (CONFIRMED, ON_REQUEST, CANCELLED, etc.).

6.7 Modificar reserva (si el proveedor lo permite)

```
POST /v1/bookings/{bookingId}/modify
```

Request (cambio de fechas como ejemplo):

```
{
"newCheckIn": "2026-03-11",
"newCheckOut": "2026-03-13",
"reason": "Customer request",
"idempotencyKey": "<uuid>"
}
```

6.8 Cancelar reserva

```
POST /v1/bookings/{bookingId}/cancel
```

Request:

```
{
"reasonCode": "CUSTOMER_REQUEST",
"comment": "Cancelacion solicitada por el cliente",
"refundMethod": "ORIGINAL_PAYMENT",
"idempotencyKey": "<uuid>"
}
```

6.9 Mensajeria / requests al hotel

```
POST /v1/bookings/{bookingId}/messages
```

Sirve para enviar solicitudes especiales o coordinar temas operativos.

```
{
"channel": "EMAIL",
"subject": "Solicitud especial",
"message": "Por favor confirmar early check-in si es posible.",
"language": "es"
}
```

6.10 Pagos y conciliacion

Endpoints para registrar pagos externos, consultar transacciones y hacer conciliacion (especialmente para CREDIT_BALANCE y VCC).

```
POST /v1/bookings/{bookingId}/payments
GET /v1/bookings/{bookingId}/payments
GET /v1/settlements?from=YYYY-MM-DD&to=YYYY-MM-DD
GET /v1/transactions?from=YYYY-MM-DD&to=YYYY-MM-DD
```

7. Webhooks (eventos)

Usa webhooks para recibir cambios de estado asincronos: confirmaciones tardias, cancelaciones por el proveedor, cambios de precio, estatus de pago, etc.

```
POST /v1/webhooks
GET /v1/webhooks
DELETE /v1/webhooks/{webhookId}
```

Registro de webhook (ejemplo):

```
{
"url": "https://tu-dominio.com/webhooks/cloudrooms",
"events": [
"booking.created",
"booking.confirmed",
"booking.cancelled",
"booking.modified",
"payment.succeeded",
"payment.failed"
],
"secret": "<random_shared_secret>"
```

```
}
```

Validacion: firma cada webhook con HMAC-SHA256 usando el secret compartido (similar al esquema de firma de requests).

8. Errores, timeouts y reintentos

Codigos HTTP:

Codigo	Uso
200/201	OK / creado
400	Request invalido (validacion)
401/403	No autorizado / firma invalida / permisos
404	No encontrado
409	Conflicto (idempotencia, estado no permite accion)
422	Regla de negocio (por ejemplo, oferta expirada)
429	Rate limit
500/502/503	Error servidor / gateway - reintentar con backoff

Reintentos: solo reintenta automaticamente en 429/5xx y en timeouts. En POST con efecto, siempre usa Idempotency-Key. Backoff recomendado: 1s, 2s, 4s, max 3-5 intentos.

9. Quickstart Node.js (axios)

Ejemplo de cliente minimo con firma HMAC y timeout:

```
import axios from "axios";
import { signRequest } from "./sign.js";
import { v4 as uuidv4 } from "uuid";

const baseURL = process.env.CLOUDROOMS_BASE_URL;
const apiKey = process.env.CLOUDROOMS_API_KEY;
const secret = process.env.CLOUDROOMS_API_SECRET;

export async function crRequest(method, path, bodyObj) {
const body = bodyObj ? JSON.stringify(bodyObj) : "";
const ts = Date.now().toString();
const sig = signRequest({ secret, timestamp: ts, method, path, body });

const headers = {
"Content-Type": "application/json",
"X-CR-API-Key": apiKey,
"X-CR-Timestamp": ts,
"X-CR-Signature": sig,
};

if (["POST","PUT","PATCH"].includes(method.toUpperCase())) {
headers["Idempotency-Key"] = uuidv4();
}

const res = await axios.request({
baseURL,
url: path,
method,
data: bodyObj || undefined,
```

```
timeout: Number(process.env.TIMEOUT_MS || 15000),
headers,
validateStatus: () => true,
});

return res;
}
```

Uso:

```
const res = await crRequest("POST", "/v1/search/availability", {
checkIn: "2026-03-10",
checkOut: "2026-03-12",
currency: "COP",
nationality: "CO",
occupancies: [{ adults: 2, children: [7] }]
});

if (res.status !== 200) throw new Error(JSON.stringify(res.data));
console.log(res.data);
```

10. Checklist de pruebas

- Autenticacion: firma valida y reloj sincronizado (NTP).
- Busqueda: /search/availability con nationality y currency (COP) devuelve offers.
- Cotizacion: /quote devuelve quoteId vigente.
- Reserva: /bookings crea booking y retorna bookingId + estado.
- Consulta: /bookings/{id} coincide con tu persistencia interna.
- Cancelacion: respeta politica; prueba reintento con Idempotency-Key.
- Pagos: prueba CUSTOMER_CARD (si aplica), VCC_PER_BOOKING, CREDIT_BALANCE y EXTERNAL_TRANSFER segun tu operacion.
- Webhooks: recibes booking.confirmed y payment.succeeded; validas firma.
- Observabilidad: logs con requestId, partnerBookingRef y bookingId.

Anexo A. Referencias (contexto HyperGuest)

HyperGuest publica, a nivel general, que ofrece API para datos dinamicos (bookable/ARI) en modo PULL y PUSH, y API para datos estaticos. Tambien describe los modelos Marketplace y Bilateral Relationship.

Referencia	URL
FAQ Travel Providers (modelos + servicios)	https://www.hyperguest.com/faqs/demand
FAQ Do you provide API/XML? (PULL/PUSH/static)	https://www.hyperguest.com/ufaq/do-you-provide-api-xml