Nivel de acabado del microservicio payments-and-suscriptions

1. Payments-and-suscriptions

Pareja: Ramón Gavira Sánchez y Miguel Encina Martínez

2. Descripción general

Este es un **microservicio secundario** de la aplicación SocialBeats, cuya responsabilidad principal es gestionar todo lo relacionado con **suscripciones y pagos**. A diferencia de los microservicios principales (beats-upload, beats-interaction, user-auth), este microservicio actúa como orquestador entre:

- **Stripe**: Pasarela de pagos para procesar transacciones y gestionar suscripciones.
- **SPACE**: Sistema de pricing de la asignatura que gestiona contratos, planes y límites de uso.

El microservicio **no gestiona un recurso REST tradicional** como los microservicios principales, sino que proporciona endpoints de integración con servicios externos de pago y gestiona el estado de las suscripciones de los usuarios mediante el modelo de `src/models/Subscription.jsx`.

3. Funcionalidades principales

3.1 Gestión de Suscripciones

- Crear sesiones de checkout para nuevas suscripciones (Stripe Checkout).
- Consultar estado de suscripción del usuario.
- Actualizar plan de suscripción (upgrades y downgrades).
- Cancelar suscripciones.
- Gestionar Add-Ons (extras al plan base).

3.2 Integración con Stripe

- Creación de Customers de Stripe.
- Sesiones de Checkout para pagos con tarjeta.
- Sesiones de Setup para añadir métodos de pago.
- Actualización de planes con prorrateo automático.
- Cancelación de suscripciones.
- Procesamiento de Webhooks para sincronizar estado.

3.3 Integración con SPACE

- Creación de contratos cuando un usuario se registra.
- Actualización de planes y add-ons en SPACE.
- Sincronización de límites de uso según el plan.
- Eliminación de contratos cuando se elimina un usuario.

3.4 Comunicación Kafka

- Escucha eventos `USER_DELETED` para eliminar suscripciones y contratos.
- Reacciona ante eliminación de usuarios para mantener consistencia.

4. Arquitectura y componentes

Estructura del proyecto

```
.
├── .github/                              # Workflows de GitHub Actions
│   └── workflows/
│       ├── conventional-commits.yml
│       ├── create-releases.yml
│       ├── linter.yml
│       └── run-tests.yml
│
├── .husky/                               # Git hooks
│   ├── commit-msg
│   └── pre-commit
│
├── docs/                                 # Documentación adicional
│   └── PRODUCTION_MIGRATION.md
│
├── scripts/
│   └── copyEnv.cjs                       # Script para copiar archivos .env
│
├── spec/
│   └── oas.yaml                          # OpenAPI Specification
│
├── src/
│   ├── db.js                             # Conexión a MongoDB
│   │
│   ├── config/
│   │   └── plans.config.js               # Configuración centralizada de planes y
add-ons
│   │
│   ├── controllers/
│   │   ├── addOnController.js            # Controlador para gestión de add-ons
│   │   └── subscriptionController.js     # Controlador principal de suscripciones
│   │
│   ├── middlewares/
│   │   ├── authMiddlewares.js            # Verificación de autenticación
│   │   ├── internalMiddleware.js         # Verificación de API key interna
│   │   └── webhookMiddleware.js          # Procesamiento de webhooks de Stripe
│   │
│   ├── models/
│   │   └── Subscription.js               # Modelo de suscripción en MongoDB
│   │
│   ├── routes/
│   │   ├── aboutRoutes.js                # Rutas de documentación
│   │   ├── addOnRoutes.js                # Rutas de add-ons
│   │   ├── healthRoutes.js               # Health check
│   │   └── subscriptionRoutes.js         # Rutas principales de suscripciones
```

```
│   │
│   ├── services/
│   │   ├── kafkaConsumer.js            # Consumidor de eventos Kafka
│   │   ├── spaceService.js             # Cliente para SPACE API
│   │   └── stripeService.js            # Cliente para Stripe API
│   │
│   └── utils/
│       ├── spaceConnection.js          # Configuración de conexión a SPACE
│       └── versionUtils.js             # Utilidades de versión
│
├── tests/
│   ├── integration/
│   │   └── subscription.flow.test.js
│   ├── mocks/
│   │   ├── spaceMock.js
│   │   └── stripeMock.js
│   ├── setup/
│   │   ├── setup.js
│   │   ├── testEnv.js
│   │   └── testHelpers.js
│   └── unit/
│       ├── about.test.js
│       ├── addons.test.js
│       ├── auth.test.js
│       ├── controller.mocked.test.js
│       ├── health.test.js
│       ├── plans.config.test.js
│       ├── spaceService.test.js
│       ├── stripeService.test.js
│       ├── subscription.model.test.js
│       ├── subscription.test.js
│       └── webhook.test.js
│
├── main.js                             # Entry point de la aplicación
├── logger.js                           # Logger centralizado
├── Dockerfile                          # Imagen de producción
├── Dockerfile-dev                      # Imagen de desarrollo
├── docker-compose.yml
├── docker-compose-dev.yml
└── package.json
```

Persistencia (MongoDB)

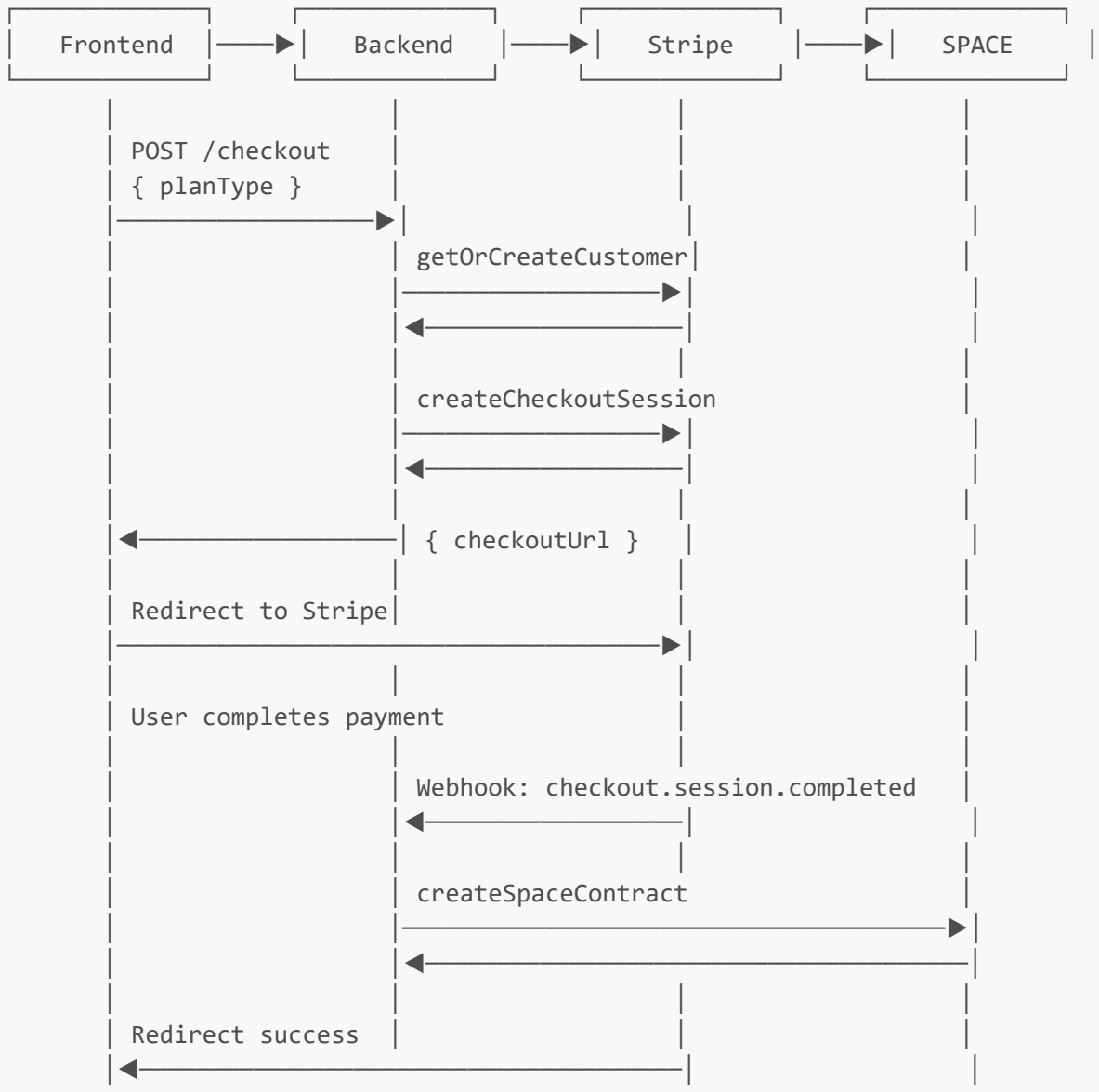
El modelo `Subscription` almacena:

```
{
  userId: String,              // ID del usuario (string, no ObjectId)
  username: String,
  email: String,
  stripeCustomerId: String,    // ID del customer en Stripe
  stripeSubscriptionId: String, // ID de la suscripción en Stripe
```

```
  stripePriceId: String,       // Price ID del plan actual
  status: String,              // active, canceled, past_due, incomplete,
trialing, unpaid
  planType: String,            // FREE, PRO, STUDIO
  activeAddOns: [{             // Add-ons activos
    name: String,
    stripeSubscriptionItemId: String,
    status: String,
    activatedAt: Date
  }],
  currentPeriodStart: Date,
  currentPeriodEnd: Date,
  cancelAtPeriodEnd: Boolean,
  metadata: Object             // Info adicional (downgrades pendientes, etc.)
}
```

5. Integración con Stripe

5.1 Flujo de Checkout (Nueva Suscripción)



5.2 Implementación en código

Creación de Customer (`stripeService.js`):

```
export const getOrCreateCustomer = async (email, metadata = {}) => {
  // Buscar customer existente por email
  const existingCustomers = await stripe.customers.list({ email, limit: 1 });

  if (existingCustomers.data.length > 0) {
    return existingCustomers.data[0];
  }

  // Crear nuevo customer
  return await stripe.customers.create({ email, metadata });
};
```

Sesión de Checkout (`stripeService.js`):

```
export const createCheckoutSession = async ({
  customerId, priceId, successUrl, cancelUrl, metadata = {}
}) => {
  return await stripe.checkout.sessions.create({
    customer: customerId,
    payment_method_types: ['card'],
    mode: 'subscription',
    line_items: [{ price: priceId, quantity: 1 }],
    success_url: successUrl,
    cancel_url: cancelUrl,
    metadata,
    subscription_data: { metadata },
    allow_promotion_codes: true,
  });
};
```

5.3 Flujo de Upgrade/Downgrade

Upgrade (cobro inmediato con prorrateo):

```
Usuario PRO → STUDIO
1. updateSubscriptionPlan con prorationBehavior: 'create_prorations'
2. Stripe cobra la diferencia prorrateada inmediatamente
3. Usuario tiene acceso inmediato al nuevo plan
4. Se actualiza SPACE con el nuevo plan
```

Downgrade (cambio diferido al final del periodo):

```
Usuario STUDIO → PRO
1. Se crea un SubscriptionSchedule en Stripe
2. Fase 1: Mantener plan STUDIO hasta fin de periodo
3. Fase 2: Cambiar a PRO al inicio del siguiente periodo
4. Se guarda metadata.pendingPlanChange en la suscripción
5. El webhook subscription_schedule.completed actualiza SPACE
```

5.4 Gestión de Add-Ons

Los Add-Ons se implementan como **Subscription Items** adicionales en Stripe:

```
// Añadir AddOn
export const addSubscriptionItem = async ({ subscriptionId, priceId }) => {
  return await stripe.subscriptionItems.create({
    subscription: subscriptionId,
    price: priceId,
    quantity: 1,
    proration_behavior: 'create_prorations',
  });
};

// Eliminar AddOn
export const removeSubscriptionItem = async (subscriptionItemId) => {
  return await stripe.subscriptionItems.del(subscriptionItemId, {
    proration_behavior: 'create_prorations',
  });
};
```

6. Webhooks de Stripe

6.1 Configuración

El middleware de webhooks debe montarse **ANTES** de `express.json()` porque Stripe necesita el body en formato raw para verificar la firma:

```
// main.js
app.post(
  '/api/v1/payments/webhook',
  webhookMiddleware,           // express.raw() + guarda rawBody
  verifyStripeSignature,       // Verifica firma de Stripe
  subscriptionController.handleWebhook
);

app.use(express.json());       // Después del webhook
```

6.2 Eventos procesados

Evento	Acción
`checkout.session.completed`	Crear suscripción en DB y contrato en SPACE
`customer.subscription.updated`	Sincronizar estado y plan
`customer.subscription.deleted`	Crear plan FREE automáticamente
`invoice.paid`	Confirmar renovación exitosa
`invoice.payment_failed`	Marcar como `past_due`
`subscription_schedule.completed`	Aplicar downgrade diferido

6.3 Verificación de firma

```
export const verifyWebhookSignature = (payload, signature) => {
  // En producción: verificar firma
  if (webhookSecret && !webhookSecret.includes('dummy')) {
    return stripe.webhooks.constructEvent(payload, signature, webhookSecret);
  }

  // En desarrollo: permitir sin verificación
  logger.warn('⚠ Webhook signature verification DISABLED (development mode)');
  return JSON.parse(payload.toString());
};
```

7. Integración con SPACE

7.1 Operaciones disponibles

Crear contrato (`spaceService.js`):

```
export const createSpaceContract = async ({ userId, username, plan, addOns = {} })
=> {
  // Si existe, actualizar; si no, crear nuevo
  await spaceClient.contracts.addContract({
    userContact: { userId, username },
    billingPeriod: { autoRenew: true, renewalDays: 30 },
    contractedServices: { [SPACE_SERVICE_NAME]: SPACE_SERVICE_VERSION },
    subscriptionPlans: { [SPACE_SERVICE_NAME]: plan },
    subscriptionAddOns: addOns,
  });
};
```

Actualizar contrato:

```
export const updateSpaceContract = async ({ userId, plan, addOns = {} }) => {
  await spaceClient.contracts.updateContractSubscription(userId, {
```

```
    contractedServices: { socialbeats: '1.0' },
    subscriptionPlans: { socialbeats: plan },
    subscriptionAddOns: addOns,
  });
};
```

Cancelar contrato (downgrade a FREE):

```
export const cancelSpaceContract = async (userId) => {
  await spaceClient.contracts.updateContractSubscription(userId, {
    contractedServices: { socialbeats: '1.0' },
    subscriptionPlans: { socialbeats: 'FREE' },
    subscriptionAddOns: {},
  });
};
```

Eliminar contrato:

```
export const deleteSpaceContract = async (userId) => {
  await fetch(`${SPACE_URL}/api/v1/contracts/${userId}`, {
    method: 'DELETE',
    headers: { 'x-api-key': SPACE_API_KEY },
  });
};
```

7.2 Sincronización con el plan

Cada vez que se actualiza el plan en Stripe (vía webhook o API), se sincroniza con SPACE:

1. **checkout.session.completed** → `createSpaceContract()`
2. **customer.subscription.updated** → `updateSpaceContract()`
3. **customer.subscription.deleted** → `cancelSpaceContract()` (downgrade a FREE)

8. Comunicación con Kafka

8.1 Eventos consumidos

El microservicio escucha el topic `users-events`:

```
await consumer.subscribe({ topic: 'users-events', fromBeginning: true });
```

Evento `USER_DELETED`:

Cuando un usuario se elimina del sistema (desde `user-auth`), se ejecuta:

```
case 'USER_DELETED':
  const userId = data.userId;

  // 1. Buscar todas las suscripciones del usuario
  const subscriptions = await Subscription.find({ userId });

  // 2. Cancelar cada suscripción en Stripe
  for (const subscription of subscriptions) {
    if (subscription.stripeSubscriptionId) {
      await
stripeService.cancelSubscriptionImmediately(subscription.stripeSubscriptionId);
    }
    // 3. Eliminar de la base de datos
    await Subscription.deleteOne({ _id: subscription._id });
  }

  // 4. Eliminar contrato en SPACE
  await spaceService.deleteSpaceContract(userId);
```

8.2 Circuit Breaker en Kafka

El consumidor implementa reintentos con backoff:

```
const MAX_RETRIES = 5;
const RETRY_DELAY = 5000;      // 5 segundos
const COOLDOWN_AFTER_FAIL = 30000;  // 30 segundos

while (true) {
  try {
    await consumer.connect();
    await producer.connect();
    // ... suscribirse y escuchar
    break;
  } catch (err) {
    if (attempt >= MAX_RETRIES) {
      // Cooldown y reiniciar contador
      await sleep(COOLDOWN_AFTER_FAIL);
      attempt = 1;
    } else {
      attempt++;
      await sleep(RETRY_DELAY);
    }
  }
}
```

8.3 Dead Letter Queue (DLQ)

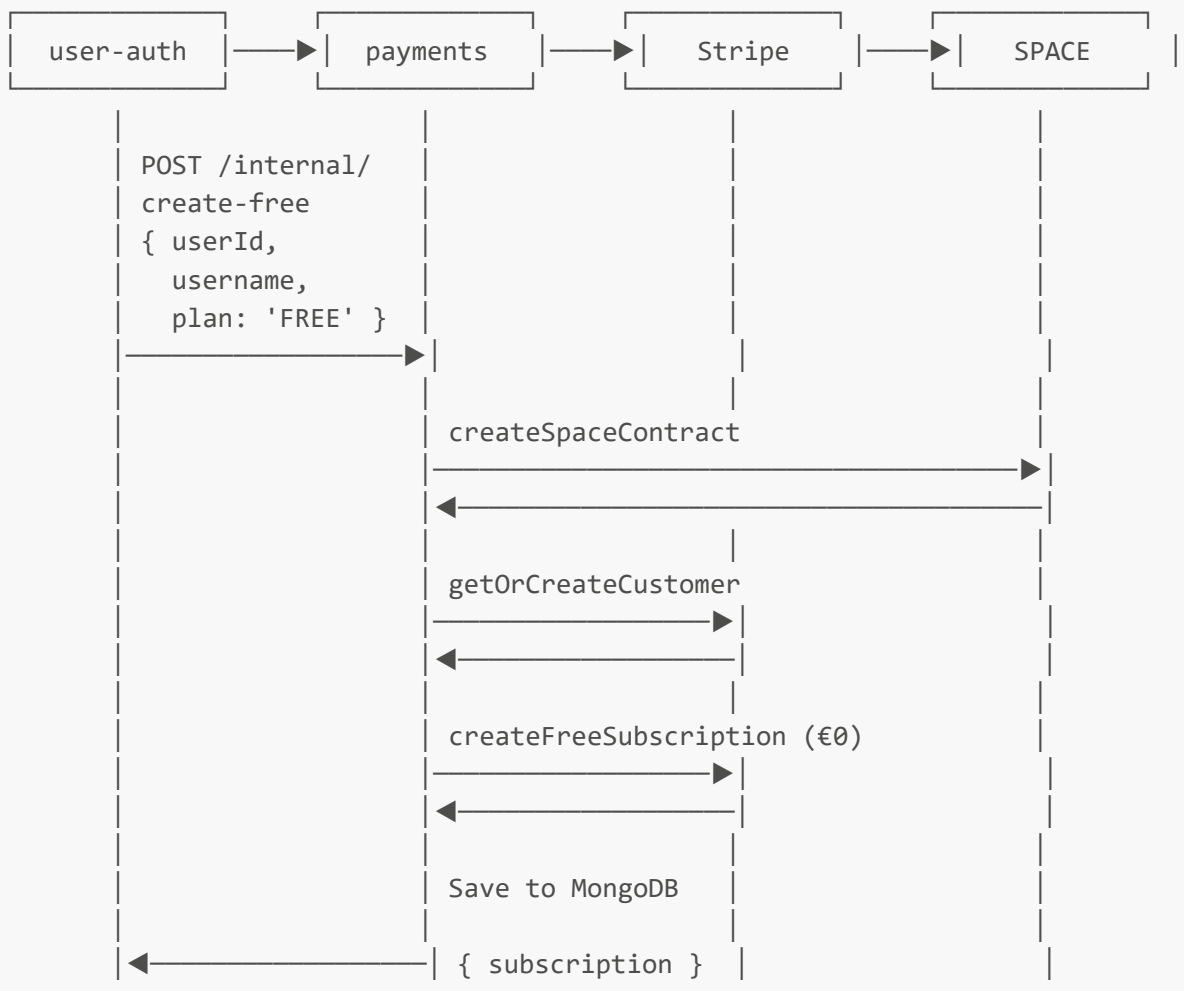
Los eventos que fallan se envían a una DLQ para análisis:

```
async function sendToDLQ(event, reason) {
  await producer.send({
    topic: 'payments-interaction-dlq',
    messages: [{
      value: JSON.stringify({
        originalEvent: event,
        error: reason,
        timestamp: new Date().toISOString(),
      }),
    }],
  });
}
```

9. Creación de suscripción gratuita al registrar usuario

9.1 Flujo completo

Cuando un usuario se registra en la aplicación, el microservicio `user-auth` hace una llamada interna a este endpoint para crear automáticamente un plan gratuito:



9.2 Endpoint interno

```
// POST /api/v1/payments/internal/create-free-contract
// Header: x-internal-api-key: <INTERNAL_API_KEY>

export const createFreeContract = async (req, res) => {
  const { userId, username, plan, email } = req.body;

  // 1. Crear contrato en SPACE
  await spaceService.createSpaceContract({ userId, username, plan });

  // 2. Crear Customer en Stripe
  const customer = await stripeService.getOrCreateCustomer(email, { userId,
username });

  // 3. Crear suscripción gratuita en Stripe (precio €0, sin tarjeta)
  const stripeSubscription = await stripeService.createFreeSubscription({
    customerId: customer.id,
    priceId: freePriceId,
    metadata: { userId, username, planType: 'FREE' },
  });

  // 4. Guardar en MongoDB
  const subscription = await Subscription.findOneAndUpdate(
    { userId },
    {
      userId, username, email,
      planType: 'FREE',
      status: 'active',
      stripeCustomerId: customer.id,
      stripeSubscriptionId: stripeSubscription.id,
      ...
    },
    { upsert: true, new: true }
  );

  res.status(200).json({ subscription });
};
```

9.3 Middleware de autenticación interna

Este endpoint solo es accesible con una API key interna:

```
// internalMiddleware.js
export const requireInternalApiKey = (req, res, next) => {
  const apiKey = req.headers['x-internal-api-key'];

  if (!apiKey || apiKey !== process.env.INTERNAL_API_KEY) {
    return res.status(401).json({
      error: 'UNAUTHORIZED',
      message: 'Invalid or missing internal API key',
    });
  }
```

```
  next();
};
```

10. Configuración de planes

10.1 Definición centralizada

Los planes están definidos en `src/config/plans.config.js`:

```
export const PLANS = {
  FREE: {
    name: 'FREE',
    price: 0.00,
    stripePriceId: process.env.STRIPE_PRICE_FREE,
    features: {
      advancedProfile: true,
      banner: false,
      maxBeats: 3,
      maxBeatSize: 10, // MB
      maxStorage: 30,  // MB
      maxPlaylists: 1,
      // ...
    },
  },
  PRO: {
    name: 'PRO',
    price: 9.99,
    stripePriceId: process.env.STRIPE_PRICE_PRO,
    features: {
      banner: true,
      cover: true,
      maxBeats: 30,
      maxBeatSize: 25,
      maxStorage: 750,
      maxPlaylists: 10,
      // ...
    },
  },
  STUDIO: {
    name: 'STUDIO',
    price: 29.99,
    stripePriceId: process.env.STRIPE_PRICE_STUDIO,
    features: {
      decoratives: true,
      downloads: true,
      privatePlaylists: true,
      maxBeats: Infinity,
      maxStorage: 1000,
      // ...
    },
```

```
  },
};
```

10.2 Add-Ons disponibles

```
export const ADDONS = {
  decoratives: {
    name: 'decoratives',
    displayName: 'Decoratives Pack',
    price: 2.99,
    stripePriceId: process.env.STRIPE_PRICE_ADDON_DECORATIVES,
    availableForPlans: ['PRO'], // Solo disponible para PRO
  },
  promotedBeat: {
    name: 'promotedBeat',
    displayName: 'Promoted Beat',
    price: 4.99,
    stripePriceId: process.env.STRIPE_PRICE_ADDON_PROMOTED,
    availableForPlans: ['PRO', 'STUDIO'],
  },
  // ...
};
```

11. API REST

Endpoints principales

Método	Endpoint	Descripción
POST	/api/v1/payments/checkout	Crear sesión de checkout
GET	/api/v1/payments/subscription	Obtener estado de suscripción
PUT	/api/v1/payments/subscription	Actualizar plan (upgrade/downgrade)
DELETE	/api/v1/payments/subscription	Cancelar suscripción
POST	/api/v1/payments/subscription/complete-upgrade	Completar upgrade tras añadir método de pago
POST	/api/v1/payments/addons	Añadir add-on
DELETE	/api/v1/payments/addons/:addonName	Eliminar add-on
GET	/api/v1/payments/addons	Listar add-ons activos
POST	/api/v1/payments/webhook	Webhook de Stripe
POST	/api/v1/payments/internal/create-free-contract	Crear contrato FREE (interno)

Documentación

- OpenAPI Spec: `spec/oas.yaml`
- Swagger UI: `/api/v1/docs/`

12. Tests

El proyecto incluye tests unitarios y de integración:

- `tests/unit/stripeService.test.js` - Tests del servicio de Stripe
- `tests/unit/spaceService.test.js` - Tests del servicio de SPACE
- `tests/unit/subscription.test.js` - Tests de controladores
- `tests/unit/webhook.test.js` - Tests de procesamiento de webhooks
- `tests/unit/addons.test.js` - Tests de add-ons
- `tests/integration/subscription.flow.test.js` - Tests de flujo completo

Ejecución:

```
npm test              # Tests unitarios
npm run test:coverage # Con cobertura
```