💳 STEP 3: Subscription & Payment System - API Documentation

📋 Overview

Complete Razorpay integration with subscription management, webhooks, and feature gating.

🔑 Base URL

```
http://localhost:3000/api
```

📦 PLANS API

1. Get All Plans (Public)

GET `/plans`

Get all available subscription plans.

Response (200 OK):

```json
```

```
[
  {
   "id": "uuid",
   "name": "STARTER",
   "displayName": "Starter",
   "description": "Perfect for small NGOs",
   "price": 0,
   "currency": "INR",
   "interval": "MONTHLY",
   "trialDays": 14,
   "features": {
     "basic_features": true,
     "email_support": true
   },
   "maxProjects": 3,
   "maxUsers": 3,
   "maxBeneficiaries": 50,
   "maxStorage": 512,
   "financeEnabled": false,
   "complianceEnabled": false,
   "apiAccess": false,
   "customBranding": false,
   "isActive": true
  },
  {
   "id": "uuid",
   "name": "GROWTH",
   "displayName": "Growth",
   "price": 2999,
   "currency": "INR",
   "interval": "MONTHLY",
   "maxProjects": 10,
   "maxUsers": 10,
   "maxBeneficiaries": 500,
   "financeEnabled": true
   // ... more fields
  }
]
```

---

2. Get Plan by ID (Public)

GET `/plans/:id`

Get specific plan details.

Response (200 OK):

```json
{
  "id": "uuid",
  "name": "GROWTH",
  "displayName": "Growth",
  "price": 2999,
  // ... plan details
}
```

💳 SUBSCRIPTIONS API

3. Get Current Subscription

GET `/subscriptions/current`

Get tenant's current subscription details.

Headers:

```
Authorization: Bearer {accessToken}
```

Response (200 OK):

```
json
```

```json
{
 "id": "uuid",
 "tenantId": "uuid",
 "planId": "uuid",
 "status": "ACTIVE",
 "razorpaySubscriptionId": "sub_xyz123",
 "currentPeriodStart": "2024-01-01T00:00:00Z",
 "currentPeriodEnd": "2024-02-01T00:00:00Z",
 "plan": {
  "id": "uuid",
  "name": "GROWTH",
  "displayName": "Growth",
  "price": 2999,
  "maxProjects": 10,
  "maxUsers": 10
 },
 "payments": [
  {
   "id": "uuid",
   "amount": 2999,
   "status": "SUCCESS",
   "paymentDate": "2024-01-01T10:00:00Z"
  }
 ],
 "invoices": [
  {
   "id": "uuid",
   "invoiceNumber": "INV-202401-0001",
   "total": 2999,
   "status": "PAID"
  }
  ]
}
```

Status Values:

- `TRIAL` - Free trial period
- `ACTIVE` - Active subscription
- `PAST_DUE` - Payment failed, grace period
- `CANCELLED` - Subscription cancelled
- `EXPIRED` - Subscription expired

---

4. Create Subscription

POST `/subscriptions`

Create new subscription for tenant.

Headers:

```
Authorization: Bearer {accessToken}
```

Request Body:

```json
{
  "planId": "uuid",
  "isYearly": false
}
```

Response (201 Created):

```json
{
  "subscription": {
    "id": "uuid",
    "tenantId": "uuid",
    "planId": "uuid",
    "status": "ACTIVE",
    "razorpaySubscriptionId": "sub_xyz123",
    "currentPeriodStart": "2024-01-01T00:00:00Z",
    "currentPeriodEnd": "2024-02-01T00:00:00Z"
  },
  "paymentLink": "https://rzp.io/l/abc123",
  "razorpaySubscriptionId": "sub_xyz123"
}
```

Business Logic:

1. Checks if tenant already has active subscription
2. Creates Razorpay plan (if needed)
3. Creates Razorpay subscription
4. Returns payment link for user
5. User completes payment on Razorpay
6. Webhook activates subscription

Free Plans:

- No payment link returned

- Subscription activated immediately

---

5. Update Subscription (Upgrade/Downgrade)

PUT `/subscriptions`

Change subscription plan.

Headers:

```
Authorization: Bearer {accessToken}
```

Request Body:

```json
{
  "newPlanId": "uuid"
}
```

Response (200 OK):

```json
```

```
{
 "subscription": {
  "id": "uuid",
  "planId": "new-plan-uuid",
  "status": "ACTIVE"
 },
 "paymentLink": "https://rzp.io/l/xyz456"
}
```

Business Logic:

1. Cancels current Razorpay subscription
2. Creates new subscription with new plan
3. Prorates charges (handled by Razorpay)
4. Returns new payment link

---

6. Cancel Subscription

POST `/subscriptions/cancel`

Cancel subscription.

Headers:

```
Authorization: Bearer {accessToken}
```

Request Body:

```
json
```

```
{
  "cancelAtPeriodEnd": true
}
```

Response (200 OK):

```json
{
  "message": "Subscription will be cancelled at the end of current period",
  "cancelAt": "2024-02-01T00:00:00Z"
}
```

Options:

- `cancelAtPeriodEnd: true` - Cancel at end of billing period
- `cancelAtPeriodEnd: false` - Cancel immediately

💰 PAYMENTS API

7. Get Payment History

GET `/payments/history`

Get all payments for tenant.

Headers:

```
Authorization: Bearer {accessToken}
```

Response (200 OK):

```json
[
  {
    "id": "uuid",
    "subscriptionId": "uuid",
    "amount": 2999,
    "currency": "INR",
    "status": "SUCCESS",
    "razorpayPaymentId": "pay_xyz123",
    "razorpayOrderId": "order_abc123",
    "paymentMethod": "card",
    "paymentDate": "2024-01-01T10:00:00Z",
    "invoice": {
      "id": "uuid",
      "invoiceNumber": "INV-202401-0001",
      "pdfUrl": "https://..."
    }
  }
]
```

Payment Statuses:

- `PENDING` - Payment initiated
- `PROCESSING` - Payment being processed
- `SUCCESS` - Payment successful
- `FAILED` - Payment failed
- `REFUNDED` - Payment refunded

8. Get Invoices

GET `/payments/invoices`

Get all invoices for tenant.

Headers:

```
Authorization: Bearer {accessToken}
```

Response (200 OK):

```json
```

```json
[
  {
    "id": "uuid",
    "subscriptionId": "uuid",
    "paymentId": "uuid",
    "invoiceNumber": "INV-202401-0001",
    "amount": 2999,
    "tax": 539.82,
    "total": 3538.82,
    "currency": "INR",
    "status": "PAID",
    "dueDate": "2024-01-01T00:00:00Z",
    "paidAt": "2024-01-01T10:00:00Z",
    "billingPeriodStart": "2024-01-01T00:00:00Z",
    "billingPeriodEnd": "2024-02-01T00:00:00Z",
    "pdfUrl": "https://..."
  }
]
```

---

9. Razorpay Webhook (Internal)

POST `/payments/webhook`

Webhook endpoint for Razorpay events.

⚠️ **PUBLIC ENDPOINT** - No authentication required 🔒 **SIGNATURE VERIFICATION** - Validates webhook signature

Headers:

```
x-razorpay-signature: {signature}
```

Request Body: (varies by event)

```json
{
  "event": "subscription.charged",
  "payload": {
    "payment": { "entity": { /* payment data */ } },
    "subscription": { "entity": { /* subscription data */ } }
  }
}
```

Supported Events:

- `subscription.activated` - Subscription activated
- `subscription.charged` - Payment successful
- `subscription.completed` - Subscription ended
- `subscription.cancelled` - Subscription cancelled
- `subscription.paused` - Subscription paused
- `subscription.resumed` - Subscription resumed
- `payment.authorized` - Payment authorized
- `payment.captured` - Payment captured
- `payment.failed` - Payment failed
- `invoice.paid` - Invoice paid

Response (200 OK):

```json
{
  "success": true
}
```

Webhook Flow:

1. Razorpay sends event to webhook
2. Verify signature (CRITICAL!)
3. Process event based on type
4. Update database
5. Send notifications
6. Return success

🚀 Integration Flow

Complete Subscription Flow

```
1. User selects plan on frontend
   ↓
2. Frontend calls POST /subscriptions
   ↓
3. Backend creates Razorpay subscription
   ↓
4. Backend returns payment link
   ↓
5. Frontend redirects user to Razorpay
```

```
   ↓
6. User completes payment
   ↓
7. Razorpay sends webhook (payment.authorized)
   ↓
8. Backend updates subscription status
   ↓
9. Razorpay sends webhook (subscription.charged)
   ↓
10. Backend creates payment & invoice
   ↓
11. Backend sends confirmation email
   ↓
12. User can access platform
```

🎯 Feature Gating

Using @Feature Decorator

typescript

```
import { Controller, Get } from '@nestjs/common';
import { Feature } from '../../common/guards/feature-gate.guard';

@Controller('finance')
export class FinanceController {

  // This route requires 'finance' feature
  @Get('reports')
  @Feature('finance')
  async getFinanceReports() {
    // Only accessible if plan has financeEnabled: true
  }
}
```

Available Features:

- `finance` - Financial module access
- `compliance` - Compliance features
- `api` - API access
- `branding` - Custom branding

Error Response (403 Forbidden):

```json
{
  "statusCode": 403,
  "message": "This feature is not available in your current plan. Please upgrade to access finance."
}
```

📊 Usage Limits

Using @UsageLimit Decorator

```typescript
import { Controller, Post } from '@nestjs/common';
import { UsageLimit } from '../../common/guards/usage-limit.guard';

@Controller('projects')
export class ProjectsController {

  // Check project limit before creation
  @Post()
  @UsageLimit('projects')
  async createProject() {
    // Only accessible if under maxProjects limit
  }
}
```

Available Limits:

- `projects` - maxProjects from plan
- `users` - maxUsers from plan
- `beneficiaries` - maxBeneficiaries from plan

Error Response (403 Forbidden):

```json
```

```
{
  "statusCode": 403,
  "message": "You have reached the maximum limit of 10 projects for your plan. Please upgrade to add more."
}
```

🧪 Testing

Test with Razorpay Test Mode

1. **Get Test API Keys:**
   - Login to Razorpay Dashboard
   - Switch to Test Mode
   - Get Test Key ID & Secret
2. **Test Cards:**

```
Success: 4111 1111 1111 1111
CVV: Any 3 digits
Expiry: Any future date
```

3. **Test Flow:**

```bash
```

```
# 1. Get plans
curl http://localhost:3000/api/plans

# 2. Create subscription (login first)
curl -X POST http://localhost:3000/api/subscriptions \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"planId":"PLAN_ID","isYearly":false}'

# 3. Use payment link from response
# Complete payment on Razorpay

# 4. Check subscription status
curl http://localhost:3000/api/subscriptions/current \
  -H "Authorization: Bearer YOUR_TOKEN"
```

🔐 Security Considerations

Webhook Security

```
typescript
```

```
// ALWAYS verify webhook signature
const isValid = razorpay.verifyWebhookSignature(
  JSON.stringify(payload),
  signature
);

if (!isValid) {
  throw new BadRequestException('Invalid signature');
}
```

Best Practices

1. ✅ Always verify webhook signatures
2. ✅ Use HTTPS in production
3. ✅ Store Razorpay secrets securely
4. ✅ Log all webhook events
5. ✅ Handle idempotency (duplicate webhooks)
6. ✅ Implement retry logic
7. ✅ Send email confirmations
8. ✅ Monitor failed payments

💡 Business Logic

Trial Management

- New tenants get 14-day free trial (configurable)
- Trial ends automatically after period

- Tenant can subscribe anytime during trial
- After trial, account suspended until payment

Billing Cycles

- **Monthly**: Billed every 30 days
- **Yearly**: Billed annually (2 months free)
- Auto-renewal enabled by default
- Can cancel anytime

Failed Payments

1. Payment fails → Subscription status: `PAST_DUE`
2. 3-day grace period (configurable)
3. Send payment reminder emails
4. After grace period → Suspend account
5. Can retry payment anytime

Prorated Charges

- Handled automatically by Razorpay
- Upgrade: Charged difference immediately
- Downgrade: Credit applied to next cycle

📈 Usage Tracking

Daily Snapshots

- Automated cron job runs at midnight
- Records usage for all active tenants
- Metrics: projects, users, beneficiaries, storage

Limit Warnings

- Automated cron job runs at 9 AM
- Checks if tenants near limits (80%+)
- Sends warning emails
- Suggests plan upgrade

🆘 Error Handling

Common Errors

1. Already has subscription

```json
{
  "statusCode": 400,
  "message": "Tenant already has an active subscription"
}
```

2. Plan not found

```json
{
  "statusCode": 404,
  "message": "Plan not found or inactive"
}
```

3. Feature not available

```json
{
  "statusCode": 403,
  "message": "This feature is not available in your current plan"
}
```

4. Usage limit exceeded

```json
{
  "statusCode": 403,
  "message": "You have reached the maximum limit of 10 projects"
}
```

✅ Testing Checklist

- [ ] Can view all plans (public)

- [ ] Can get specific plan details
- [ ] Can create subscription with payment
- [ ] Can create free subscription (Starter plan)
- [ ] Payment link redirects to Razorpay
- [ ] Webhook signature verification works
- [ ] Subscription activates after payment
- [ ] Invoice generated after payment
- [ ] Can view payment history
- [ ] Can view invoices
- [ ] Can upgrade subscription
- [ ] Can downgrade subscription
- [ ] Can cancel subscription (end of period)
- [ ] Can cancel subscription (immediate)
- [ ] Feature gating blocks unauthorized access
- [ ] Usage limits enforced correctly
- [ ] Failed payment updates status to PAST_DUE
- [ ] Grace period works correctly
- [ ] Usage tracking records metrics
- [ ] Super admin bypasses limits

🚀 Next Steps

STEP 3 Complete! ✅

Ready for **STEP 4**:

- Project Management Module
- Beneficiary Management

- Donor Management
- Fund Allocation

Reply with **"Proceed to STEP 4"** when ready!