Shothik Payment Service - Complete System Overview

Developer: Shaikot Kundu Akash
Team: Shothik AI
Purpose: Centralized Payment, Credit & Subscription Management

---

📘 Table of Contents

17. RabbitMQ Events

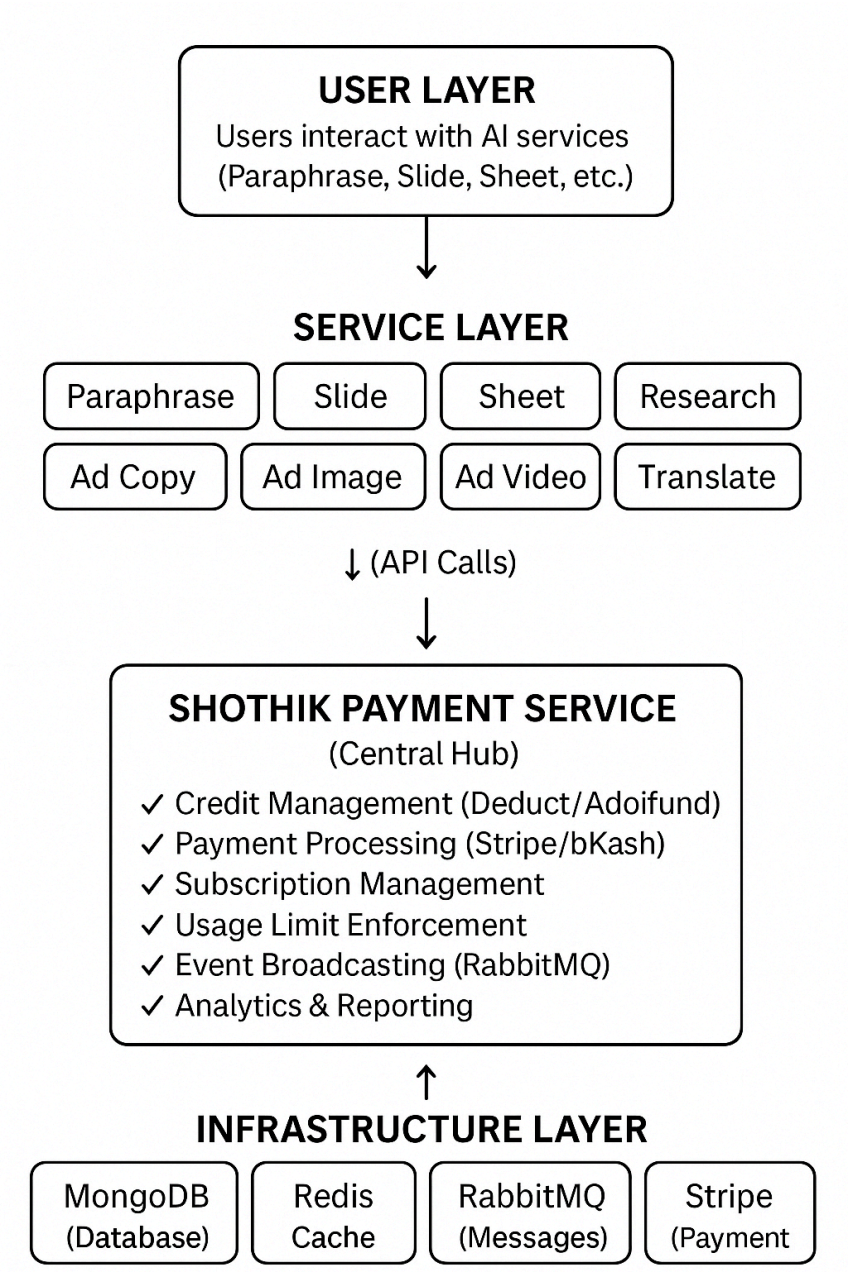
18. Service List & Credit Costs

19. System Metrics

20. Security Architecture Layers

21. Version & Credits

📊 System at a Glance



Details : System Architecture.txt

🔑 Integration Summary

Integration Type	Count	Purpose
REST API Endpoints	9	Credit, balance, and payment operations
Outgoing Events	13	Asynchronous notifications
Incoming Events	0	Payment Service only publishes
API Keys	1 per service	Auth per microservice
Total Integration Points	**22**	Complete communication matrix

---

📡 Complete Communication Flow

1️⃣ API Communication (Synchronous)

SERVICE → PAYMENT SERVICE

Endpoints

```
POST /deduct                        → Deduct user credits
GET  /balance/:userId                 → Check credit balance
POST /reverse                       → Reverse transaction
GET  /subscriptions/:userId/validate   → Check access limits
```

Authentication

Authorization: ApiKey sk_yourservice_xyz...

2️⃣ Event Communication (Asynchronous)

PAYMENT SERVICE → SERVICES *(via RabbitMQ)*
Exchange: `payment.events (Topic)`

🧩 Events Published by Payment Service

🔸 Service Control Events (Critical – Must Listen)

Event	Description
service.*.stop_user	Block user access
service.*.resume_user	Unblock user access
usage.limit_exceeded	User exceeded limit
usage.limit_reset	Limit has been reset

🔸 Credit Events (Optional – For Analytics)

Event	Description
credit.deducted	Credits used
credit.added	Credits purchased
credit.refunded	Credits refunded

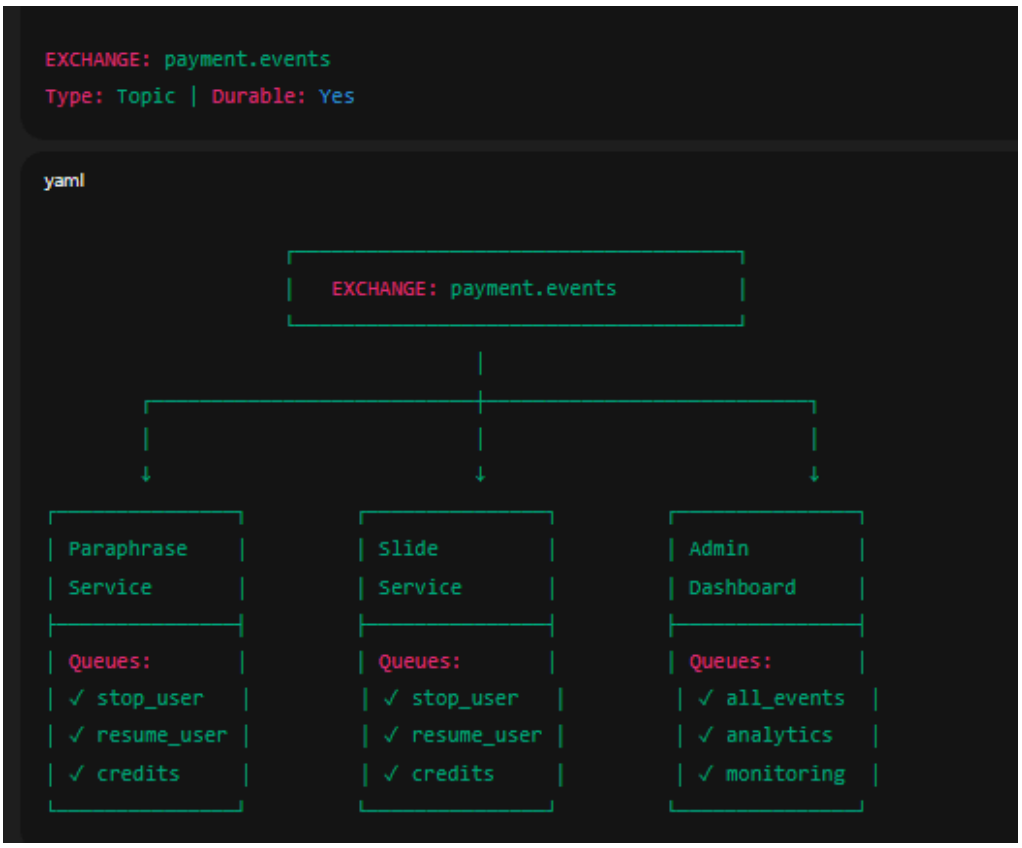
🔸 Payment Events (Optional – For Tracking)

Event	Description
payment.completed	Payment successful
payment.failed	Payment failed
payment.refunded	Payment refunded

🔸 Subscription Events (Optional – For Marketing/Admin)

Event	Description
subscription.created	New subscription
subscription.changed	Plan upgraded/downgraded
subscription.cancelled	Subscription ended
subscription.expired	Subscription expired

🏗️ RabbitMQ Channel Architecture



Property	Value
Exchange	payment.events
Type	Topic
Durable	Yes

Queue Naming Convention:
`{service_name}.{event_type}`
Example: `paraphrase_service.stop_user`

Routing Key Pattern:
`{category}.{subject}.{action}`
Example: `service.paraphrase_service.stop_user`

🔄 Real-World Usage Scenarios

Scenario 1: Normal Request Processing

1. User sends request → Paraphrase Service
2. Service checks blocklist ✅
3. `GET /balance/:userId → { balance: 100 }`
4. `POST /deduct → { transactionId: 'txn_123', newBalance: 95 }`
5. Process success ✅
6. Return response: **Credits Used: 5 | Remaining: 95**

Scenario 2: Processing Error (with Reversal)

1. Credits deducted
2. AI service fails ❌
3. `POST /reverse → { success: true, newBalance: 100 }`
4. Return message: *Processing failed, credits refunded.*

Scenario 3: User Limit Exceeded

1. User balance = 0
2. Payment Service triggers `service.paraphrase_service.stop_user`
3. Service adds user to blocklist
4. Next request → **403 Forbidden – Service suspended**

Scenario 4: User Purchases Credits

1. User pays $10 = 100 credits via Stripe
2. Payment Service verifies and adds credits
3. Publishes events:
    - `payment.completed`
    - `credit.added`
    - `service.*.resume_user`
4. All services remove user from blocklist ✅

📈 Statistics & Metrics

Event Type	Volume/Day	Priority
credit.deducted	~10,000	Normal
service.*.stop_user	~50	Critical
service.*.resume_user	~50	Critical
payment.completed	~100	High
subscription.*	~20	Normal

Endpoint	Calls/Day	Avg Response Time
/deduct	~10,000	50ms
/balance/:userId	~5,000	20ms
/reverse	~100	40ms
/validate-access	~15,000	30ms

🛡️ Security Architecture

Layer 1: API Key Authentication

Authorization: ApiKey sk_paraphrase_xyz...

- Validates API key
- Rate limits per service
- Identifies calling microservice

Layer 2: JWT Authentication

Authorization: Bearer eyJhbGc...

- Used for user-facing endpoints
- Verifies signature, expiry, and identity

Layer 3: Redis Caching & Deduplication

Key: idempotency:{requestId}
TTL: 10 minutes

- Prevents duplicate deductions
- Caches responses
- Enforces rate limits

📦 Technology Stack

Component	Technology	Purpose
API Server	Node.js + Express	REST endpoints
Database	MongoDB	Persistent data
Cache	Redis	Performance, deduplication
Message Broker	RabbitMQ	Event distribution
Payment Gateway	Stripe + bKash	Payment processing
Authentication	JWT + API Keys	Security
Container	Docker + Compose	Deployment

🎯 Developer Quick Start

1️⃣ Get API Key
`sk_paraphrase_abc123xyz...`

2️⃣ Setup Environment

```
PAYMENT_SERVICE_URL=http://localhost:3001/api/v1
PAYMENT_SERVICE_API_KEY=sk_yourservice_xyz...
SERVICE_NAME=paraphrase_service
RABBITMQ_URL=amqp://admin:shothik_rabbitmq_2025@localhost:5672
```

3️⃣ Install Dependencies

```
npm install axios amqplib
```

4️⃣ Implement Integration
(see `QUICK_START_EXAMPLE.js`)

5️⃣ Test Integration

```
node test-payment-connection.js
node test-event-listener.js
```

🔍 Monitoring & Debugging

Event Monitor

cd shothik-central-payment/test-scripts
node monitor-events-simple.js

RabbitMQ UI

URL: http://localhost:15672
Username: `admin`
Password: `shothik_rabbitmq_2025`

CLI

rabbitmqctl list_queues
rabbitmqctl list_bindings

✅ Integration Checklist

☑ API Key obtained
☑ Environment configured
☑ Payment client implemented
☑ Event consumer implemented
☑ Blocklist handling added
☑ Credit deduction integrated
☑ Error reversal implemented
☑ Event listeners started
☑ Tests passing
☑ Monitoring setup

📡 REST API Endpoints

(9 Total — all secured via `ApiKey` or `JWT`)

1️⃣ Credit Deduction

POST /api/v1/deduct
Authorization: ApiKey sk_{service}_abc123xyz...
Content-Type: application/json

Request:

```
{
  "userId": "string",
  "amount": 5,
  "service": "paraphrase_service",
  "feature": "text_paraphrase"
}
```

Response:

```
{
  "success": true,
  "transactionId": "txn_abc123xyz",
  "newBalance": 95,
  "previousBalance": 100
}
```

2️⃣ Balance Check

```
GET /api/v1/balance/:userId
Authorization: ApiKey sk_{service}_abc123xyz...
```

Response:

```
{
  "success": true,
  "userId": "user_123",
  "balance": 100,
  "currency": "credits"
}
```

3️⃣ Credit Reversal

```
POST /api/v1/reverse
Authorization: ApiKey sk_{service}_abc123xyz...
```

Response:

```
{
  "success": true,
  "amountReversed": 5,
  "newBalance": 100
}
```

4️⃣ Access Validation

GET /api/v1/subscriptions/:userId/validate-access
Authorization: ApiKey sk_{service}_abc123xyz...

Response:

```
{
  "success": true,
  "canAccess": true,
  "subscription": {
    "plan": "pro",
    "status": "active"
  }
}
```

5️⃣ Payment Initiation

POST /api/v1/payments/initiate
Authorization: Bearer {jwt_token}

Response:

```
{
  "success": true,
  "sessionId": "session_abc123",
  "paymentUrl": "https://checkout.stripe.com/..."
}
```

6️⃣ Stripe Webhook

`POST /api/v1/webhooks/stripe`

7️⃣ bKash Webhook

`POST /api/v1/webhooks/bkash`

8️⃣ User Wallet

GET /api/v1/wallet
Authorization: Bearer {jwt_token}

Response:

```
{
  "success": true,
  "wallet": { "balance": 150, "status": "active" }
}
```

9️⃣ Subscription Creation

POST /api/v1/subscriptions
Authorization: Bearer {jwt_token}

Response:

```
{
  "success": true,
  "subscription": { "plan": "pro", "status": "active" }
}
```

🔐 Admin Panel – API Key Management

Key Format:
`sk_{service_name}_{random_32_chars}`
Example: `sk_paraphrase_a8f9d2e3c4b5a6f7e8d9c0b1a2f3e4d5`

Part	Description
`sk_`	Prefix
`{service_name}`	Service identifier
`{random_string}`	32-char secure key

Admin Endpoints Overview

#	Endpoint	Method	Purpose
1	/admin/api-keys/generate	POST	Generate new key
2	/admin/api-keys	GET	List all keys
3	/admin/api-keys/:keyId	GET	Get key details
4	/admin/api-keys/:keyId	PATCH	Update key
5	/admin/api-keys/:keyId/rotate	POST	Rotate key
6	/admin/api-keys/:keyId	DELETE	Revoke key
7	/admin/api-keys/:keyId/usage	GET	Get usage
8	/admin/api-keys/bulk	POST	Bulk operations
9	/admin/api-keys/:keyId/audit-log	GET	Audit log
10	/admin/api-keys/:keyId/validate	POST	Validate key health

🐰 RabbitMQ Events

Exchange	payment.events
Type	topic
Durable	true

Published Events (13 Total)

#	Event Type	Priority	Purpose
1	service.stop_user	Critical	Notify user suspension
2	service.resume_user	Critical	Resume access
3	credit.deducted	Normal	Track usage
4	credit.added	High	Credit added
5	credit.refunded	High	Refund issued
6	payment.completed	High	Successful payment
7	payment.failed	High	Failed payment

8	payment.refunded	High	Payment refunded
9	subscription.created	Normal	Subscription created
10	subscription.changed	Normal	Plan changed
11	subscription.cancelled	Normal	Cancelled
12	subscription.expired	Normal	Expired
13	usage.limit_exceeded	High	Limit alert

💳 Service List & Credit Costs

Service	Key Prefix	Function	Credit Cost
Paraphrase	sk_paraphrase_	Text rewriting	1 / 100 chars
Slide Generation	sk_slide_	Presentation creation	10 / deck
Sheet Generation	sk_sheet_	Data analysis	5 / sheet
Deep Research	sk_research_	In-depth analysis	20 / query
Ad Copy Text	sk_adcopy_	Ad copy generation	3 / copy
Ad Image	sk_adimage_	AI image creation	15 / image
Ad Video	sk_advideo_	Video creation	50 / video
Summarize	sk_summarize_	Text summarization	2 / 1000 words
Translate	sk_translate_	Language translation	1 / 200 chars

📊 System Metrics

Event	Avg/Day	Peak Time (UTC+6)
credit.deducted	~10,000	2–5 PM
payment.completed	~100	10 AM–2 PM
subscription.created	~20	Various

Endpoint	Daily Calls	Avg Time	Error Rate
/deduct	10,000	50 ms	0.5%
/balance/:userId	5,000	20 ms	0.1%
/reverse	100	40 ms	2%
/validate-access	15,000	30 ms	0.3%
/payments/initiate	150	500 ms	5%

🛡️ Security Architecture Layers (Summary)

1. **API Key Authentication** – microservice identity
2. **JWT Authorization** – frontend & users
3. **Rate Limiting** – 1000 req/hr/service
4. **Idempotency Control** – prevent duplication via Redis

🧾 Version & Credits

System: Shothik Payment Service
Version: 2.0
Developer: Shaikot Kundu Akash
Team: Shothik AI
Last Updated: October 2025
