Stripe and Twilio Integration: With Tenants (Multi-Tenant)

Prepared for Product, Engineering and Finance. Stack: React + Node.js (Express) + MongoDB. Prices are public list prices as of November 2026 and must be verified against each vendor's pricing page before commitments. Items marked (verify) change often.

1. Summary

In a multi-tenant app, each customer organisation (tenant) needs its own payments and SMS behaviour, yet the platform owner wants one codebase, one control point and clear cost tracking per tenant.

Service	Recommended tenant model	Who pays the vendor	Who earns margin
Stripe (payments)	Connect Express accounts if tenants collect money from their own customers; one Stripe Customer per tenant if tenants simply pay you a SaaS subscription	Tenant pays Stripe fees (Connect) or you pay and recover via price (SaaS)	Platform via application fee or subscription price
Twilio (SMS)	One subaccount per tenant under one parent account, with an optional bring-your-own-account path for enterprise tenants	Platform (single consolidated invoice)	Platform, via resale price minus Twilio cost

Golden rules for all three:

1. React never holds secrets. It only calls your Node API.
2. Every collection and every vendor call carries a tenant_id.
3. Webhooks are the source of truth for payment, delivery and account state.
4. Keep your own ledger of usage per tenant and reconcile it against vendor data daily.

2. Reference Architecture

```
React (browser) -> Node/Express API -> Stripe / Twilio API
                        |   ^
                        v   +---- webhooks (Stripe, Twilio status)
                    MongoDB (tenants, vendor accounts, usage ledgers)
```

Suggested core collections (MongoDB):

Collection	Key fields	Indexes
tenants	name, plan, status	Unique on name or slug
stripe_accounts	tenant_id, stripe_account_id, stripe_customer_id, charges_enabled, payouts_enabled	Unique on tenant_id
twilio_accounts	tenant_id, account_sid, api_key_sid, api_secret (encrypted), mode (subaccount or byo)	Unique on tenant_id
message_log	tenant_id, channel, vendor_id, segments, status, cost, created_at	Compound on tenant_id and created_at; unique on vendor_id
usage_daily	tenant_id, day, category, count, cost	Unique compound on tenant_id, day and category (allows upsert)
wallet_ledger	tenant_id, amount, type, ref, created_at	Compound on tenant_id and created_at
processed_events	event_id	Unique on event_id, with a TTL index to expire old events

MongoDB notes:

1. Put tenant_id first in every compound index and in every query filter, so tenants never read each other's documents.
2. Encrypt secrets (API secrets, refresh tokens) with MongoDB field-level encryption or an application-level KMS wrapper, never as plain fields.

3. Update wallet balances with atomic operators such as $inc, or use multi-document transactions (requires a replica set or Atlas) when a debit and a ledger entry must succeed together.
4. Use upserts on usage_daily so the nightly Twilio sync can safely re-run.
5. Use a unique index on processed_events.event_id to make webhook handling idempotent.
6. Use the official MongoDB Node driver or Mongoose; both fit the Express backend.

A small provider layer (for example getStripeForTenant, getTwilioClientForTenant) returns the right credentials for a tenant_id, so business code never cares which model a tenant uses.

3. Stripe With Tenants

3.1 Choose the pattern first

Pattern	Use when	Stripe objects
A. Stripe Connect	Tenants sell to their own customers and you take a commission or fee (marketplace, booking, school or clinic fees)	One connected account per tenant
B. Platform billing	Tenants only pay you for the software	One Customer and one Subscription per tenant on your own Stripe account
C. Both	Tenants pay you a subscription and also collect from their customers	Customer plus connected account per tenant

3.2 Procedure for Pattern A (Connect)

1. Enable Connect in the Stripe Dashboard and complete the platform profile.
2. At tenant onboarding, create an Express connected account and store its ID against tenant_id.
3. Create an Account Link and redirect the tenant admin to Stripe's hosted onboarding. Stripe collects identity, bank and tax details, so you do not build KYC forms.

4. Listen to account.updated on a Connect webhook and cache charges_enabled and payouts_enabled. Never trust the return URL alone.
5. For each sale, create a Checkout Session on the server with the tenant's account as the destination and your application fee. Never accept prices or account IDs from the browser.
6. Handle checkout.session.completed, refunds and disputes on the platform webhook, skipping event IDs already processed.
7. Give tenants a login link to their Express dashboard for payouts and tax documents.

APIs to use for tenant-level Stripe Connect

Layer	API or library	Purpose
Backend (Node, stripe package)	Accounts API: accounts.create (type express, tenant_id in metadata), accounts.retrieve, accounts.update	Create the tenant's connected account, read its status, change payout settings
Backend	Account Links API: accountLinks.create (type account_onboarding)	One-time onboarding URL for the tenant admin
Backend	Login Links API: accounts.createLoginLink	Opens the tenant's Express dashboard
Backend	Checkout Sessions API: checkout.sessions.create with application_fee_amount and transfer_data.destination	Take a payment, keep your fee, send the rest to the tenant
Backend	Refunds API: refunds.create with reverse_transfer and refund_application_fee	Refund and claw back the tenant share and your fee
Backend	Balance Transactions and Payouts APIs	Per-tenant reconciliation of fees and payouts
Backend	Webhooks: webhooks.constructEvent on two endpoints (platform and Connect). Events: account.updated, checkout.session.completed, charge.refunded, charge.dispute.created, payout.failed	Source of truth for tenant and payment state

Layer	API or library	Purpose
Backend	Your own REST routes, for example POST /api/stripe/onboard, GET /api/stripe/status, POST /api/checkout	What React calls; each route checks the logged-in tenant
Frontend (React)	No Stripe secret APIs. Redirect the browser to the Account Link URL and the Checkout URL returned by your routes	Hosted onboarding and payment
Frontend, optional	Connect embedded components (@stripe/connect-js and @stripe/react-connect-js); the backend adds accountSessions.create	Onboarding and payouts inside your own React pages
Frontend, optional	@stripe/stripe-js and @stripe/react-stripe-js (Payment Element); the backend uses the PaymentIntents API	Custom card form instead of hosted Checkout

3.3 Procedure for Pattern B (platform billing tenants)

1. Create Products and Prices (monthly and yearly plans) in Stripe.
2. At tenant signup, create a Stripe Customer with metadata tenant_id and store the ID.
3. Start a Checkout Session in subscription mode, or use the Customer Portal for plan changes and card updates.
4. Handle invoice.paid, invoice.payment_failed and customer.subscription.updated webhooks to set the tenant's plan and status (active, past_due, suspended).
5. For usage-based charges (for example SMS credits), report usage through Stripe Billing meters or create invoice items from your usage_daily table at the end of each period.

APIs to use for platform billing of tenants (Pattern B)

Layer	API or library	Purpose
Backend (stripe package)	Customers API: customers.create and customers.update (tenant_id in metadata)	One Stripe Customer per tenant

Layer	API or library	Purpose
Backend	Checkout Sessions API in subscription mode	Start a tenant's paid plan
Backend	Billing Portal API: billingPortal.sessions.create	Tenant self-service for cards, plan changes and cancellation
Backend	Subscriptions API: retrieve, update, cancel	Plan changes and tenant offboarding
Backend	Invoices API: invoices.list and invoices.retrieve	Invoice history shown in your app
Backend	Billing Meters API (billing.meterEvents.create) or Invoice Items API (invoiceItems.create)	Charge usage such as SMS credits from your usage_daily table
Backend	Webhook events: invoice.paid, invoice.payment_failed, customer.subscription.updated, customer.subscription.deleted	Set tenant plan and status (active, past_due, suspended)
Frontend (React)	Redirect to the Checkout URL and the Portal URL returned by your routes; call your own routes such as GET /api/billing/plan and GET /api/billing/invoices	Plan and invoice screens without handling card data

3.4 Billing and cost handling

Item	Typical list cost (verify)	Who bears it
Card processing	About 2.9% + $0.30 per successful online card payment in the US. India and other countries differ	Connect: your choice (platform or tenant). Platform billing: you

Item	Typical list cost (verify)	Who bears it
Connect Express	About $2 per monthly active connected account plus about 0.25% + $0.25 per payout (verify; varies by country)	Platform, usually recovered from the application fee or plan price
Stripe Billing	Roughly 0.5% to 0.8% of recurring invoice volume on top of processing (verify)	Platform
Disputes	About $15 per dispute, plus the disputed amount	Platform is typically liable on destination charges; recover from the tenant by contract
Currency conversion and cross-border payout	Extra percentage fees	Whoever you assign in the fee model

Worked example: a customer pays $100 to a tenant with a 10% platform fee. Stripe takes about $3.20 in processing from the platform. Platform keeps $10.00 - $3.20 = $6.80, the tenant receives $90.00. If you instead want the tenant to bear processing cost, raise the fee or deduct processing from the transfer. Decide this before launch.

Tenant-wise visibility: Stripe reports per connected account (balance, payouts, fees). Save the payment intent ID, amount, application fee and tenant_id on every order so your own reports match.

3.5 Pros and cons

Pros	Cons
Stripe handles KYC, bank payouts and tenant dashboards (Connect)	Platform carries refund and dispute exposure
Tenant money is separated from platform money	Rules and fees differ by country; cross-border payouts are limited
Automatic fee splitting; no manual payouts	Two webhook endpoints and more states to test
Subscriptions, invoices and tax tools are built in (Billing)	Billing adds a per-invoice percentage on top of card fees

Pros	Cons
Per-tenant revenue reporting	Connect onboarding can stall if a tenant has incomplete documents

4. Twilio With Tenants

4.1 Models

Model	Description	Best for
Subaccount per tenant	One parent Twilio account owned by you; each tenant gets a subaccount (own SID, numbers, logs, usage)	Standard and self-service tenants
Bring your own account	Tenant owns a separate Twilio account and gives you an API key	Enterprise or regulated tenants who need their own contract

Twilio bills only the parent account, as one consolidated invoice. Tenant-wise numbers come from the Usage Records API per subaccount and from your own message_log. Tenant billing is your responsibility.

4.2 Procedure (subaccount model)

1. Upgrade the parent account, enable billing and two-factor authentication, and create a parent API key.
2. At tenant onboarding, call the Accounts API to create a subaccount and store its SID.
3. Buy or assign a phone number and a Messaging Service inside that subaccount. Point status and inbound webhooks to your API with the tenant ID in the URL.
4. Collect the tenant's business details in your own React form and submit regulatory registration (US A2P 10DLC brand and campaign, India DLT, toll-free verification). Block sending until approved.
5. Send through a single backend endpoint that checks wallet balance and daily limits, then calls Twilio with that tenant's subaccount.
6. Validate X-Twilio-Signature on every webhook with the subaccount's Auth Token (signatures use the Auth Token, not the API secret).

7. Log every message immediately, update status from callbacks, and reconcile cost nightly.
8. Offboard by suspending, then closing the subaccount after exporting needed data.

APIs to use for tenant-level Twilio

Layer	API or library	Purpose
Backend (Node, twilio package)	Accounts API: api.v2010.accounts.create and update (status suspended or closed)	Create, suspend and close a tenant subaccount
Backend	API Keys API (newKeys.create)	Optional key per subaccount instead of one parent key
Backend	Available Phone Numbers and Incoming Phone Numbers APIs	Search, buy and configure numbers and webhook URLs per tenant
Backend	Messaging Services API (messaging.v1.services)	Sender pool per tenant and link numbers to it
Backend	Messages API: messages.create and messages.fetch with statusCallback	Send SMS and read delivery state
Backend	Trust Hub API (customer profiles, end users, supporting documents)	Tenant business identity for regulatory registration
Backend	A2P 10DLC APIs (brand registrations and US A2P campaigns under messaging.v1)	US brand and campaign registration. India DLT is registered through the operator DLT portal and linked in Twilio (verify)
Backend	Usage Records API (usage.records.daily)	Per-tenant daily counts and cost for reconciliation
Backend	Usage Triggers API (usage.triggers.create)	Spend alerts or pause per subaccount
Backend	Balance API (balance.fetch)	Balance checks for bring-your-own accounts

Layer	API or library	Purpose
Backend	Webhook signature validation (twilio.validateRequest or the webhook middleware)	Verify every status and inbound webhook
Backend	Your own routes, for example POST /api/messages, POST /twilio/status/:tenantId, POST /twilio/inbound/:tenantId, GET /api/usage	What React and Twilio call
Frontend (React)	No Twilio SDK needed for SMS. Forms post tenant business details and consent settings to your routes; screens read message history, usage and wallet from your routes	Compliance onboarding, send screen, usage dashboard

4.3 SMS usage tracking

1. Real-time ledger: insert into message_log at send time with segment count (price is not known yet).
2. Update status via the status callback.
3. Nightly job reads Usage Records per subaccount and upserts into usage_daily.
4. Compare the sum of daily cost against your ledger and alert on drift.
5. Use Usage Triggers per subaccount to alert or pause on abnormal spend.

Twilio keeps message logs for a limited period (verify current retention), so keep your own copy for audits, consent proof and tenant dashboards.

4.4 Billing and cost handling

Item	Typical list cost (verify)	Handling
US SMS segment	About $0.0083 outbound, plus carrier fee (about $0.0035 on AT&T), so about $0.0118 all-in	Platform pays Twilio; resells to tenant
US MMS	About $0.022 outbound plus higher carrier fees	Same

Item	Typical list cost (verify)	Handling
Inbound SMS	About $0.0083	Pass through or bundle
Failed message processing	About $0.001 per failed message	Absorb or pass through
10DLC brand and campaign	Brand about $4.50 to $46 one-time; campaign vetting about $15; monthly campaign fee about $1.50 to $10	Pass to tenant in plan price
Phone number rental	Varies by number type and country	Pass to tenant with margin
India SMS	Per Twilio India pricing page; DLT registration required	Verify before quoting

Worked example: cost $0.0118 per segment, resale $0.0160, margin $0.0042 (about 26%). At 100,000 segments a month, tenant pays $1,600, Twilio bills you $1,180, margin is about $420. Across 100 such tenants that is about $42,000 per month before registration costs, bad debt and operating cost.

Use prepaid wallets. Estimate cost at send time (segments times your rate), then correct from the nightly Usage Records. Auto-pause tenants at zero balance. This protects you from unpaid usage and fraud.

4.5 Pros and cons

Pros	Cons
One invoice and one set of top-level credentials	Spam from one tenant can put the whole parent account at risk
Instant, API-driven onboarding	You carry unpaid-usage, fraud and carrier-fine exposure
Per-tenant isolation of numbers, logs and keys	You must build and run tenant compliance onboarding
Resale margin and volume-discount leverage	Subaccount caps and shared throughput limits (verify)
Central suspend control	Enterprise tenants may want their own Twilio contract

5. Cost Handling Summary

Service	Vendor invoices	Per-tenant record	Tenant billed by	Margin source
Stripe Connect	Tenant (processing) and platform (platform fees)	Per connected account and your orders table	Stripe takes your fee at payment time	Application fee
Stripe Billing	Platform	Customer and invoice per tenant	Stripe invoices on your account	Subscription price
Twilio subaccounts	Platform only, consolidated	Usage Records per subaccount plus message_log	Your wallet or invoice	Resale markup
Twilio BYO	Tenant	Tenant console; display-only via API key	Twilio bills the tenant	Optional platform fee

Monthly P&L per tenant (illustrative): subscription revenue + application fees + SMS margin - Stripe Billing and Connect account fees - number rental and registration costs - bad debt and fraud reserve - support and engineering cost.

6. Risks and Controls

1. Abuse: per-tenant rate limits, STOP handling, consent logs, content screening, instant suspend switch.
2. Unpaid usage: prepaid wallet, auto-pause, spend alerts.
3. Credential leakage: secrets manager or KMS, rotation, no Auth Tokens in logs, least-privilege API keys.
4. Webhook security: signature validation, HTTPS only, idempotent handlers, tenant ID in the route.
5. Data isolation: every query filters by tenant_id; add automated tests that try cross-tenant reads.
6. Compliance: terms and acceptable-use policy that give you the right to suspend; keep brand, campaign and consent records auditable.

7. Vendor limits: track subaccount count and Stripe country support.

7. Build Order

1. Tenant table, provider layer, encryption helper.
2. Stripe: Customer plus subscription for tenants, then Connect if tenants collect money.
3. Twilio: subaccount provisioning, sending, webhooks, message_log.
4. Nightly usage sync, reconciliation alerts, wallet and auto-pause.
5. Compliance onboarding forms (10DLC, DLT).
6. Tenant dashboards for usage, wallet and invoices.
7. Optional bring-your-own-account path for enterprise tenants.

8. Items to Verify Before Committing

1. Current Stripe fees for your countries, Connect pricing and Billing percentages.
2. Twilio list prices, subaccount limit, retention period and ISV program eligibility.
3. Tax handling on resale invoices (for example GST in India).
4. Countries and message types in scope (US 10DLC, India DLT).