

Courtly – Stripe Connect Integration Spec (for Cursor/Copilot)

This document is a step-by-step build plan for adding **Stripe Connect (Express)** to Courtly so club admins can collect payments for **court reservations** and **club memberships**. Use this as a Cursor/Copilot task guide.

0) Assumptions & Stack

- **Frontend:** React + TypeScript (Next.js or CRA). Auth via Firebase Auth.
 - **Backend:** Firebase Cloud Functions (Node.js/TS) + Firestore.
 - **Payments:** Stripe Connect (Express) with Checkout + Webhooks.
 - **Multitenancy:** Each Club is a Firestore `clubs/{clubId}` document.
 - **Roles:** `userType` supports `club_admin` and `member`.
-

1) Environment & Config

Create project-level env vars (CI + local):

```
STRIPE_SECRET_KEY=sk_live...
STRIPE_WEBHOOK_SECRET=whsec...
STRIPE_PUBLISHABLE_KEY=pk_live...
PLATFORM_FEE_BASIS_POINTS=300 # 3% example
APP_BASE_URL=https://courtly.app
```

For test mode, use separate `_TEST` keys or different `.env` file.

Add runtime config to Functions and Frontend (provide a typed config module the app imports).

2) Firestore Schema (new/updated fields)

Collection: `clubs/{clubId}` - `name: string` - `stripeAccountId: string | null` (e.g., `acct_1...`) - `stripeStatus: 'unlinked' | 'onboarding' | 'active' | 'restricted' | 'pending_verification'` - `stripeOnboardingComplete: boolean` - `payoutsEnabled: boolean` (mirror from Stripe) - `chargesEnabled: boolean` (mirror from Stripe) - `supportEmail: string` - `supportPhone: string` - `statementDescriptor: string` (optional) - `country: 'US' | ...` (used for currency rules) - `currency: 'usd' | ...` - `reservationSettings: { requirePaymentAtBooking: boolean; hourlyRateCents: number; }` - `membershipPlans:`

```
Array<{ id: string; name: string; priceCents: number; interval:
'month'|'year'|'one_time'; active: boolean }> - createdAt: Timestamp - updatedAt:
Timestamp
```


```
Collection: reservations/{reservationId} - clubId: string - userId: string - courtId:
string - start: Timestamp - end: Timestamp - status:
'pending'|'confirmed'|'cancelled' - priceCents: number - currency: string -
checkoutSessionId?: string - paymentIntentId?: string - paymentStatus:
'requires_payment'|'paid'|'refunded'|'failed' - createdAt: Timestamp - updatedAt:
Timestamp
```

```
Collection: membershipSubscriptions/{subscriptionId} - clubId: string - userId: string
- planId: string - status: 'incomplete'|'active'|'past_due'|'canceled'|'unpaid' -
priceCents: number - interval: 'month'|'year'|'one_time' - currency: string -
checkoutSessionId?: string - subscriptionId?: string (Stripe) - latestInvoiceId?:
string (Stripe) - customerId?: string (Stripe) - paymentStatus:
'requires_payment'|'paid'|'refunded'|'failed' - createdAt: Timestamp - updatedAt:
Timestamp
```

```
Collection: platformSettings/sensitive - platformFeeBasisPoints: number (mirror env;
optional)
```

3) Admin Dashboard – Connect Stripe UX

Route: /dashboard/club/{clubId}/payments

UI states (derived from club doc): - **Unlinked:** Show CTA “**Connect Stripe Account**” → calls backend to create Express account + Account Link. Redirect user. - **Onboarding:** Show “**Resume Stripe Onboarding**” → refresh account link. - **Active:** Show “**Stripe Connected** ”, plus Stripe Express Dashboard link, payouts/charges status, and verification banners if restricted.

Buttons: - ConnectStripeButton - ResumeOnboardingButton - OpenStripeDashboardButton

4) Backend Functions (HTTPS + Webhooks)

4.1 Create or Fetch Express Account

Endpoint: POST /api/payments/stripe/connect/start - **Auth:** Firebase ID token. Require club_admin and ownership of clubId. - **Body:** { clubId: string } - **Logic:** 1. If club has stripeAccountId, fetch account from Stripe; else stripe.accounts.create({ type: 'express' }). 2. Persist stripeAccountId, set stripeStatus='onboarding' if new. 3. Create accountLink = stripe.accountLinks.create({ account, type: 'account_onboarding', refresh_url, return_url }). 4. Return { url: accountLink.url }.

Return URL: `${APP_BASE_URL}/dashboard/club/{clubId}/payments?connected=1` **Refresh URL:** `${APP_BASE_URL}/dashboard/club/{clubId}/payments?refresh=1`

4.2 Generate Dashboard Link (Express)

Endpoint: `POST /api/payments/stripe/connect/dashboard` - **Auth:** `club_admin`. - **Body:** `{ clubId: string }` - **Logic:** `stripe.accounts.createLoginLink(club.stripeAccountId)` → `{ url }`.

4.3 Create Checkout Session – Reservation

Endpoint: `POST /api/payments/stripe/checkout/reservation` - **Auth:** member logged-in. - **Body:** `{ clubId, reservationId }` (server looks up price/metadata) - **Logic:** - Compute amount from reservation. - Ensure `club.stripeAccountId` exists and `chargesEnabled=true`. - Create **Checkout Session** with **destination charges:** - `mode: 'payment'` - `line_items: [{ name: 'Court Reservation', amount: priceCents, currency, quantity: 1 }]` - `payment_intent_data: { application_fee_amount, transfer_data: { destination: club.stripeAccountId } }` - `success_url: ${APP_BASE_URL}/reservations/${reservationId}?success=1` - `cancel_url: ${APP_BASE_URL}/reservations/${reservationId}?canceled=1` - Store `checkoutSessionId` on reservation doc.

4.4 Create Checkout Session – Membership

Endpoint: `POST /api/payments/stripe/checkout/membership` - **Auth:** member. - **Body:** `{ clubId, planId }` - **Logic:** - Lookup plan on club doc. - If recurring, use `mode: 'subscription'`; - For one-time, `mode: 'payment'`. - Use **Connected Account** destination + application fee (for recurring, use Prices & Subscriptions in connected account or platform; choose *one* model and be consistent – see Design Notes below).

4.5 Webhooks (single handler)

Endpoint: `POST /api/payments/stripe/webhook` - **Security:** Verify with `STRIPE_WEBHOOK_SECRET`. - **Handle events:** - `checkout.session.completed` - If `metadata.type==='reservation'` → mark reservation `paymentStatus='paid'`, `status='confirmed'`, set `paymentIntentId`. - If `metadata.type==='membership'` → create/activate subscription doc, store `customerId`, `subscriptionId`. - `payment_intent.payment_failed` → mark doc `paymentStatus='failed'`. - `account.updated` → sync `chargesEnabled`, `payoutsEnabled`, `requirements.currently_due` → update `stripeStatus`. - `invoice.paid` / `invoice.payment_failed` → update membership status.

Ensure idempotency by storing processed event IDs in `stripeWebhookEvents/{eventId}`.

5) Frontend Flows

5.1 Admin → Connect Flow

1. Admin opens Payments tab.
2. If unlinked, press **Connect Stripe** → call `/connect/start` → redirect to `accountLink.url`.
3. Completes onboarding on Stripe.
4. Returns to app; frontend calls a **status poller** to fetch `clubs/{clubId}` and verify `chargesEnabled/payoutsEnabled`. Show success toast.

5.2 Member → Reservation Payment

1. Member books a reservation (pending if payment required).
2. Press **Pay Now** → call `/checkout/reservation` → redirect to Stripe Checkout.
3. On success, webhook updates Firestore; UI listens to doc changes and shows **Confirmed**.

5.3 Member → Membership Purchase

1. Member selects plan.
2. Press **Subscribe** → call `/checkout/membership`.
3. Successful checkout triggers webhook; subscription becomes **active**; UI reflects status.

6) Design Notes (Important)

6.1 Destination Charges vs Direct Charges

- **Destination charges** (recommended here): Platform creates PaymentIntent/Checkout on platform account, sets `transfer_data.destination=acct_XXX` and optional `application_fee_amount`. Simplifies reporting.
- **Direct charges**: Create on connected account. Requires per-account product/price setup; fewer platform-level fees controls.

We will use **Destination Charges** for both one-time and subscription initial payments. For recurring memberships, either: - (A) Use Stripe Subscriptions on **platform account** with **transfer schedules** to connected accounts after each invoice (requires scheduled transfers logic); or - (B) Use **Subscriptions on the connected account** with **application_fee_percent** set via `on_behalf_of` + `transfer_data`. Simpler for club accounting.

Choose (B) for simplicity: create Price/Products **once per club** in connected account when plan is created, store `priceId` on `membershipPlans[].priceId`.

6.2 Taxes & Invoices

- If clubs need sales tax/VAT, enable Stripe Tax and pass `automatic_tax: { enabled: true }` in Checkout.
- Ensure statement descriptor logic respects Stripe length constraints.

6.3 Refunds & Cancellations

- Add admin UI to refund a reservation: platform calls `stripe.refunds.create({ payment_intent })`. Also update Firestore status.
 - Membership cancellation: cancel Stripe subscription and set local status.
-

7) Security & Compliance

- Always verify Firebase ID token on backend.
 - Authorization: only club admins can call connect/dashboard endpoints for their club.
 - Verify Stripe webhooks signatures.
 - Never trust client-provided prices. Lookup on server.
 - Use Firestore rules to restrict writes to payment-related fields.
-

8) Firestore Rules (sketch)

```
match /clubs/{clubId} {
  allow read: if true;
  allow update: if request.auth != null && isClubAdmin(request.auth, clubId);
}

match /reservations/{id} {
  allow read: if resource.data.clubId in userClubs(request.auth);
  allow write: if request.auth != null && canWriteReservation(request.auth,
resource.data.clubId);
}

match /membershipSubscriptions/{id} {
  allow read, write: if request.auth != null && request.resource.data.userId ==
request.auth.uid;
}
```

(Implement `isClubAdmin`, `userClubs` with custom claims or lookup.)

9) Function Stubs (Type Signatures)

Type-only hints for Cursor; actual code to be generated by agent.

```
// connect/start
POST /api/payments/stripe/connect/start
```

```

Body: { clubId: string }
Resp: { url: string }

// connect/dashboard
POST /api/payments/stripe/connect/dashboard
Body: { clubId: string }
Resp: { url: string }

// checkout/reservation
POST /api/payments/stripe/checkout/reservation
Body: { clubId: string; reservationId: string }
Resp: { sessionId: string; url: string }

// checkout/membership
POST /api/payments/stripe/checkout/membership
Body: { clubId: string; planId: string }
Resp: { sessionId: string; url: string }

// webhook
POST /api/payments/stripe/webhook

```

10) Webhook Event Mapping

Event	Action
checkout.session.completed	Update reservation/membership status to paid/active, store paymentIntentId, customerId, etc.
payment_intent.payment_failed	Mark document failed, notify user.
account.updated	Mirror chargesEnabled, payoutsEnabled, requirements → stripeStatus.
invoice.paid	Update subscription doc to active/current.
invoice.payment_failed	Mark past_due; notify member.

Idempotency: store processed `event.id` in `stripeWebhookEvents/{id}`.

11) Admin UI Components (to build)

- `<ConnectStripeCard clubId="..." />`
- Shows current status
- Buttons: Connect / Resume / Open Dashboard
- `<PayoutStatusBadge />` - reflects `chargesEnabled` / `payoutsEnabled`.

- `<MembershipPlanEditor />` – creates products/prices in connected account, saves `priceId` to plan.
 - `<ReservationPaymentButton reservationId />` – kicks off checkout.
-

12) Testing Plan

Modes: Test + Live.

Scenarios: 1. Connect flow creates `acct_xxx`, persists to Firestore, returns onboarding link. 2. Resume onboarding regenerates link if expired. 3. Reservation payment succeeds with test card `4242 4242 4242 4242` → webhook marks paid. 4. Payment failure path marks `failed` and surfaces UI error. 5. Membership subscription (monthly) creates subscription on connected account; invoices paid/failed reflected in Firestore. 6. `account.updated` toggles to `restricted` if verification docs missing; UI shows banner with dashboard link. 7. Refund path creates Stripe refund and updates reservation doc.

13) Acceptance Criteria (checklist)

- ☐ Club admin can create or connect a Stripe Express account from Payments tab.
 - ☐ After onboarding, club shows **Stripe Connected** with charges/payouts enabled flags mirrored.
 - ☐ Members can pay for reservations via Checkout; reservations auto-confirm on paid.
 - ☐ Members can purchase memberships; subscriptions reflect status from webhooks.
 - ☐ Platform fee deducted per env-config basis points.
 - ☐ Webhooks verified and idempotent; Firestore updates are atomic and secure.
 - ☐ Admin can open Stripe Express Dashboard from app.
 - ☐ All sensitive envs come from secure config; no secrets in client.
-

14) Cursor/Copilot Task Prompts

Use these verbatim as tasks:

Task A — Backend endpoints

Implement Firebase Functions endpoints described in §4. Use Stripe SDK, verify Firebase Auth, and Firestore. Return JSON `{url}` for connect/dashboard, `{sessionId, url}` for checkout endpoints.

Task B — Webhook handler

Build `/api/payments/stripe/webhook` with signature verification and event routing per §10. Update Firestore documents idempotently. Log and capture errors.

Task C — Firestore schema & rules

Add fields per §2. Implement rules per §8 with helpers `isClubAdmin`, `userClubs`.

Task D — Admin UI

Create `ConnectStripeCard` component per §11. Wire to endpoints. Implement status polling after return from onboarding.

Task E — Reservation/Membership Checkout

Implement client actions to call checkout endpoints and redirect to `url`. Listen to Firestore to reflect status changes.

Task F — Membership Plan Prices

When admin creates/edits a plan, ensure a Product/Price exists in the **connected account** and store `priceId` on the plan. Use `stripe.products.create` / `stripe.prices.create` with `stripeAccount` header targeting `acct_xxx`.

15) Monitoring & Observability

- Log function invocations with request IDs.
- Store webhook processing results in `stripeWebhookEvents` for audit.
- Add Sentry (or similar) to Functions + Frontend for error capture.

16) Rollout Plan

1. Ship in **Test Mode** behind feature flag `payments.enabled` per club.
2. Dogfood with internal club.
3. Migrate to **Live Mode** keys and rotate webhook secret.
4. Document runbook for support (common errors + resolutions).

17) Common Errors & Fixes

- `account_link` **expired**: regenerate with `/connect/start`.
 - `chargesEnabled=false` **after onboarding**: upload verification docs in Stripe → Settings.
 - `No such price` **on membership checkout**: ensure `priceId` created in the **connected account**, not platform.
 - `Payment amount mismatch` **errors**: never take amount from client; recompute server-side.
-

18) Legal/Compliance Notes (high level)

- Include Terms/Refund policy links in Checkout Session `custom_fields` / `consent` where applicable.
- Respect PCI scope by keeping card data only on Stripe-hosted pages.

End of Spec.