Store

An in-app store with a points wallet for pay-as-you-go RCIC App add-ons: fax sends, paid AI tools, complimentary or paid training meetings, and any service-request item the platform team curates.

Up to date as of v1.29.0

CONTENTS

What the Store is

The Store is RCIC App's pay-as-you-go layer. Most of the platform is subscription-shaped (Basic is free, Premium is a flat monthly or annual fee that unlocks Premium modules), but a few capabilities are genuinely usage-based — sending a fax costs the carrier real money per page, running an AI review burns model credits, training

meetings have a real human-time cost. Rather than complicating the subscription with metered overage charges, the platform routes everything usage-based through a wallet of integer points. You top up the wallet, the wallet pays for the things you do, and the wallet's balance is visible at all times so you always know what is left.

The unit is a 'store point' (we just say 'point' colloquially). One point equals one Canadian dollar at the point of purchase — there is no fractional points anywhere in the platform, and the wallet is integer-only end-to-end. Different items inside the Store cost different point counts: a fax costs 1 point per 10 transmitted pages rounded up; an AI Service Agreement review costs 1 point per run; complimentary training meetings cost 0 points and only consume your rolling 12-month allowance; ad-hoc service items live at whatever price the platform team set on the product. The Store is universally available — every tenant on every tier sees the Store entry in the sidebar from day one.

What lives in the Store

The Store carries a curated catalogue of items the platform team publishes. The platform team — Investatech operators — controls what is for sale; tenants cannot add their own products. Today the catalogue commonly includes four kinds of things: training meetings (sit down with an Investatech operator to walk through a specific module or workflow), AI capability credits (paid AI runs like the Service Agreement AI Review), ad-hoc service requests (anything from a one-off white-glove SA audit to an admin-side migration assistance), and the fax wallet (the Fax module debits the same wallet, so topping up here funds your outbound faxes too). The Store grid is updated as new offerings are published; you see the current set on /dashboard/store with a tile per published product.

The Store is always visible

Unlike most other modules where the dashboard module-access matrix can hide the sidebar entry on a per-member basis, the Store is universally visible to every team member regardless of role or seat type. The

deliberate design choice is that anyone on the team can see the catalogue and the wallet balance even if they cannot themselves purchase — your assistant should be able to confirm 'we still have enough points to send the rest of these faxes today' without needing the purchase permission. What the team CAN'T do without the right permission is spend the wallet; that gate is separate and explicit (see the next section).

Who can spend the wallet

Spending is gated by a per-member `can_store_purchase` flag on the `company_members` row. The Owner has the flag implicitly (Owners can always spend); every non-owner member defaults to false (cannot spend). The Owner toggles each team member's permission from Settings ’ Team ’ that member ’ Module Access ’ 'Can make in-store purchases'. Flipping it on grants the member full spend rights — they can buy any product, fire any paid AI run, send faxes that debit the wallet. Flipping it off removes those rights at the next request (server-side gate; the UI hides the buy buttons too). The Owner cannot delegate the topping-up of the wallet, however — only the Owner can fire a top-up against Stripe (a purposeful constraint because the top-up is a credit-card charge that lands on the Owner's billing email).

The integer-points unit

Every monetary surface inside the Store is integer points, not dollars-and-cents. The top-up tiers map cleanly (a $10 tier credits 10 paid points, a $20 tier credits 20 paid points, etc.), the product prices are whole points, the fax cost is whole points (rounded up from a per-page calculation), the AI charges are whole points per run. There is no fractional billing inside the platform; you never see '0.5 points' or '$X.XX' on a wallet balance. The integer choice keeps the math obvious — if a product costs 5 points and the wallet has 12 points, you can afford it; if it has 4, you cannot. The cents-to-points conversion happens once at the top-up: Stripe charges your card in cents, GST/HST is calculated on the cents amount, and the integer points get credited to the wallet at the end. After that everything is integer arithmetic.

The six top-up tiers

Top-ups land in six fixed tiers — there is no custom amount option. The tier table is structured so that smaller top-ups give no bonus and larger ones earn progressively more bonus points. Bonus points are credited to the non-refundable side of the wallet (more on that in the wallet-buckets section); paid points are credited to the refundable side. The tiers are: $10 ’ 10 points (no bonus), $20 ’ 22 points (10% bonus, +2), $50 ’ 60 points (20% bonus, +10), $100 ’ 130 points (30% bonus, +30), $500 ’ 700 points (40% bonus, +200), $1000 ’ 1500 points (50% bonus, +500).

The bigger-tier bonuses are a real discount — buying $500 of points at the 700-point tier means each effective point cost about 71 cents instead of $1, which is a 29% saving on what you ultimately spend. Most tenants run on the $20 or $50 tier most of the time; the $1000 tier is sized for tenants whose volume genuinely justifies a large pre-buy (the bonus pays for itself if your monthly spend is consistently above a few hundred dollars). The tier table is locked in code — the platform team cannot dial individual tier bonuses up or down per tenant; if you talk to the platform team about volume pricing, the conversation has to be a comp grant (free points credited to the wallet), not a custom tier.

The two-bucket wallet

The wallet is internally split into two buckets: 'purchased' (refundable) and 'nonrefundable'. The split matters most at account closure — when you delete your tenant, the refundable bucket gets a Stripe refund at 95% of what you paid (5% covers Stripe processing + admin overhead) plus the proportional GST/HST you originally paid on that portion, while the non-refundable bucket is forfeited. The split also drives the order of spending: when you spend points, the platform consumes the non-refundable bucket FIRST and the purchased bucket only after non-refundable runs out. This rule maximises the refundable portion you have left at closure — the bonus points the tenant got 'free' get spent before the points they actually paid for.

On the dashboard, the wallet card surfaces three numbers: paid points (refundable on closure), bonus / earned points (non-refundable), and total points available. The total is what matters for spending — both buckets are interchangeable at point of purchase; the bookkeeping happens behind the scenes. Both buckets count toward any spending decision; you cannot 'save' the refundable bucket by spending only the non-refundable one (the platform allocates automatically). Referral credits, admin grants, and bulk-tier bonus points all land in the non-refundable bucket and follow the same nonrefundable-first allocation when spent.

Topping up the wallet

Sidebar ’ Store opens the Store home page. Owners see the 'Add points' card with the six tiers laid out as buttons. Picking a tier creates a 'pending' top-up row on the platform side, opens a Stripe Checkout session for that tier's dollar amount, and redirects you to Stripe to complete payment. Stripe Checkout collects GST/HST automatically based on your billing address (Stripe's `automatic_tax` feature handles the rate lookup); the dollar amount on the receipt is the tier amount plus tax. Card payment is handled by Stripe; 3-D Secure if the issuing bank requires it; the card data never touches the platform's servers.

On success, Stripe redirects you back to /dashboard/store/receipts/<topupId>?paid=1, and a Stripe webhook fires asynchronously to the platform confirming payment. The webhook calls a single Postgres atomic RPC that allocates the receipt number, transitions the row from pending to succeeded, writes a single ledger entry crediting the paid points to the refundable bucket and the bonus points to the non-refundable bucket, and bumps both cached balances on your company row. The whole flow is idempotent — a Stripe webhook retry against the same session returns the same receipt and does not double-credit the wallet. If the webhook is briefly delayed (rare, but possible on a Vercel cold start), the receipt page shows 'Payment is processing'; refresh once after a few seconds and the receipt finalises.

Top-up receipts (RCPT-YYYY-NNNNNN)

Every successful top-up generates a numbered receipt with the format RCPT-YYYY-NNNNNN. The sequence is global across the platform (not per-tenant), it resets each January, and each receipt is unique to the top-up row that produced it. The receipt is rendered as a proper GST tax invoice carrying Investatech Inc.'s legal name, the GST/HST registration number, the date and time of purchase, the customer (your tenant's name + billing email), a single line item ('Store credit top-up — N points'), the subtotal in dollars, the tax (GST or HST broken out by line if applicable), and the total. A 'Points credited' line below the dollar total spells out the wallet impact: 'Points credited: 60 (50 paid + 10 bonus)' for the $50 tier, for example. The receipt is a downloadable PDF and a viewable web page; the PDF is the document you would attach to an expense report or a tax filing.

The product grid

Below the wallet card, the Store home page shows a grid of every published product. Each tile carries the product title (in your interface language — products carry both English and French titles, and the renderer picks based on your locale), the optional product image, the price in points, and a 'View details' button that opens the product page at /dashboard/store/products/<slug>. The product page renders the full description (also bilingual, also locale-resolved) and a Buy button. Clicking Buy fires a single POST to /api/store/orders that does the four-step atomic dance: pre-check the wallet has enough points; create the paid order row with a snapshot of the product; debit the wallet (non-refundable bucket first); dispatch the per-fulfillment-type follow-up. The atomicity means you cannot buy what you cannot afford and the wallet never goes negative even under racing requests.

The four fulfillment types

Every published product carries a fulfillment_type field that defines what happens after the wallet is debited. Four types are supported, each shaping the post-purchase experience differently.

- service_request — generic ad-hoc service. After purchase the order sits at status `paid`, the platform sends a notification email to the Investatech operators with your tenant identity and the product details, and an operator

picks up the work asynchronously. Typical use: an SA audit, an unusual data migration, a custom workflow request. The Investatech operator advances the status from paid ’ in_progress ’ fulfilled as work happens; the dashboard surfaces those state transitions.

- unlock_entitlement — turn on a capability flag for the tenant. After purchase the order goes straight to `fulfilled` and the platform writes a row into the store_entitlements table flipping the named capability to active. The write is idempotent on the entitlement key — buying the same entitlement twice does not double-charge (the second call detects the active entitlement and short-circuits before the wallet debit). Typical use: a one-time unlock of an advanced feature that is not part of the regular tier matrix.
- book_meeting — a scheduled meeting with an Investatech operator. After purchase the order sits at status `paid`, the platform stamps a deep-link URL on the order's fulfillment_state pointing at an Investatech-tenant booking page (the underlying booking system is just regular Bookings on the Investatech tenant). The receipt page surfaces a 'Book your session' CTA that takes the purchaser to the Investatech booking page with a pre-selected service. The order stays at paid until an Investatech operator marks it fulfilled after the call. Typical use: a paid training meeting that doesn't fit the rolling complimentary allowance, a one-off consultation with the platform team about platform best practices.
- inline_action_credit — paid action surfaces inside other modules. This is the type behind every paid AI run and every fax send: there is no row in the Store catalogue for it; the wallet just gets debited inline when the action fires. Examples: the Service Agreement AI Review button on the SA detail page costs 1 point per run; every successful fax delivery debits a number of points based on the page count. The wallet integration is transparent — you see the debit show up in the Store activity log alongside top-ups and product orders.

Service-request purchases — what happens after

When you buy a service_request product, the platform's job is to get the request to a human operator at Investatech and to keep you informed as the work progresses. The dispatcher fires an admin-notification email to the Investatech operator inbox carrying your tenant name, the product slug, the purchaser's email, and the order ID. From there the operator queue handles the work asynchronously — there is no automated turnaround clock, no SLA-style deadline built into the platform; the conversation about timing is between you and the operator over email or a scheduled meeting. As the operator works, they advance the order's status from paid ’ in_progress ’ fulfilled. Each transition is visible on your /dashboard/store activity log so you can see where things stand without having to chase down email threads.

Unlock-entitlement purchases — what happens after

An unlock_entitlement purchase is the instant kind. The dispatcher writes a row into the store_entitlements table with the entitlement key from the product config and active=true, the order flips to fulfilled with a fulfilled_at timestamp, and the new capability takes effect on the next request the tenant makes. The entitlement is checked at request time by code paths that want to gate behind it — the platform reads the row via a fast cached lookup. Entitlements never expire automatically; once you buy an unlock, the capability stays on for the life of the tenant. There is no time-window limitation built into the entitlements table at MVP. If a future product needs a renewing entitlement, that is a different fulfillment type rather than a modification of this one.

Book-meeting purchases — what happens after

Buying a book_meeting product fires two related things. First, the dispatcher stamps a 'bookingDeepLinkUrl' on the order's fulfillment_state — the URL points at the Investatech tenant's public booking page with the underlying Investatech-side service pre-selected (the product's fulfillmentConfig.investatechServiceSlug tells the dispatcher which service). Second, the receipt page surfaces a prominent maroon 'Book your session' CTA that opens the deep link in a new tab so you can pick a slot on the Investatech operator's calendar. The booking itself is a regular tenant-to-tenant booking on the Investatech side — same calendar invite, same confirmation email, same

lifecycle as any other booking. Because the platform has already collected payment via the Store, the underlying Investatech service is priced at $0 (the Store points cover the cost); you do not get billed a second time at the Investatech booking page.

Until you schedule the meeting (i.e. nobody has clicked through and confirmed a booking slot yet), the order is cancellable with a full points refund. The activity log surfaces a 'Cancel and refund' button on book_meeting orders; clicking it transitions the order to cancelled, refunds the points to the wallet (back to the same buckets they came from), and updates the activity log accordingly. Once the meeting is actually booked on the Investatech calendar, the cancel button disappears — at that point the standard operator-side cancel-and-reschedule rules apply, which is a conversation between you and the Investatech operator, not a self-serve platform action. The 'before scheduling' / 'after scheduling' split exists because once a calendar slot is held, the platform considers the service to have entered the operator's commitment workflow.

Inline-action-credit charges — what they look like

Inline-action-credit is the lightweight type — there is no product tile in the Store catalogue, no Buy button, no receipt page. The wallet just gets debited inline when a paid action runs inside another module. Two such surfaces ship today. The Service Agreement AI Review at /dashboard/agreements/<id> charges 1 point per run when the licensee clicks the maroon AI Review button on the detail page; the review fires immediately and the report renders inline. The Fax module debits at delivery time: when the carrier confirms a fax was successfully transmitted, the platform charges 1 point per 10 transmitted pages, rounded up, against the wallet. Failed faxes are never charged (the wallet stays untouched). Both surfaces surface the debit in the Store activity log on the Faxes or general activity tab so you can see exactly which actions cost what.

Inline-action-credit charges fire AFTER the work succeeds, not before. If the AI Review crashes mid-run, no points are charged. If a fax transmission fails (busy line, wrong number, recipient hung up), no points are charged. The 'pay only for what you actually get' rule is enforced at the platform layer — the action route checks the wallet has

enough points BEFORE attempting the action (so a known-insufficient wallet returns a clear 402 error with a CTA to top up first), then runs the action, then charges the wallet only on success. The wallet check + the success-only charge are independent atomic operations; a race where the wallet drains between check and charge returns an honest 'insufficient at settlement time' error and the work is preserved without the charge.

Spending allocation — non-refundable first

Every points debit (a product purchase, a fax charge, an AI run) follows the same allocation rule: consume from the non-refundable bucket first, then from the purchased / refundable bucket only if the non-refundable bucket runs out. If you have 30 non-refundable points and 70 paid points and you spend 50 points, the platform debits 30 non-refundable + 20 paid; your wallet ends at 0 non-refundable + 50 paid. This rule matters because of what happens at account closure (the refundable bucket gets a Stripe refund; the non-refundable bucket is forfeited). By spending non-refundable points first, the platform maximises the amount you eventually get back if you ever decide to close. The order is not configurable — there is no 'always spend paid first' toggle. The rule is uniform across every debit surface.

The Activity panel — top-ups, purchases, faxes

Below the product grid, the Store home page carries an Activity panel split into three tabs. 'Top-ups' lists every successful top-up your tenant has ever fired, most recent first, with the dollar amount paid, the tax, the total, the points credited (paid + bonus broken out), the receipt number, and a 'View receipt' link to the per-top-up receipt page. 'Purchases' lists every order placed against a Store product (every fulfillment type except inline-action-credit) with the date, status pill, fulfillment type, price in points, and a contextual action button (Cancel and refund for cancellable book_meeting orders before scheduling; Book the meeting for paid book_meeting orders that have not yet been scheduled). 'Faxes' lists every wallet debit fired by the Fax module with the date, the redacted destination number, the destination country, the page count, the points charged, and an 'Open Sent Faxes' link that jumps you straight into the Fax module's Sent log for full details.

The receipt page

Each top-up has a dedicated receipt page at /dashboard/store/receipts/<topupId>. The page is reachable from the Activity panel's 'View receipt' link AND from the post-checkout Stripe redirect (with `?paid=1` in the URL on first arrival). The receipt page renders the full tax-invoice layout: Investatech Inc.'s name and address, the GST/HST registration number, the date, the customer (your tenant), the order line item ('Store credit top-up — N points'), the subtotal, tax, total, the points-credited summary line, and a 'Download PDF' button to save a copy. The web page itself is a perpetually-available receipt — there is no expiry on viewing it, and the URL is stable so you can forward it to your accountant or attach it to expense documentation.

Comp grants and referral credit

Two other paths besides paid top-ups can credit your wallet. The platform team can issue comp grants — Investatech operators can grant free points to your wallet from the admin console, useful for incident credit, beta-tester rewards, contracted-volume arrangements, or first-month welcome bonuses. Comp grants always land in the non-refundable bucket (they were never paid for, so they cannot be refunded on closure). The grant lands instantly on your activity log with the grant reason visible (no surprise credits — the operator must type

a reason on grant). The referral program is the other path: when someone signs up for the platform with your referral cookie active and they complete their first paid top-up, you get a referral credit (typically 10 points) into the non-refundable bucket. The referral credit is automatic and visible on your activity log alongside top-ups with a 'referral_earned' source tag. There is no cap on referrals per tenant; the platform may add abuse-protection rules over time but the v1 mechanism is open.

What happens to the wallet at account closure

When you delete your tenant account, the platform sweeps the wallet through a closure-refund calculation. The unspent paid points (the refundable bucket) get a Stripe refund: the platform calculates the dollar value (1 paid point equals $1), takes 95% of that value (5% covers Stripe's per-transaction fees and the admin overhead), adds the proportional GST/HST you originally paid on that exact dollar amount, and fires a Stripe refund against the original payment intent that funded those points. The wallet is then zeroed out and a final ledger row is written for the audit trail. Non-refundable points (bonus, referral_earned, comp grants) are forfeited at closure — no refund is made for those because no money ever changed hands for them in the first place.

Two edge cases shape the actual outcome. If exactly one Stripe PaymentIntent underlies the unspent paid points (the tenant did one big top-up and spent some of it), the platform fires the refund against that PI automatically and the funds land on the original card within Stripe's normal refund window (3-10 business days). If multiple PaymentIntents underlie the unspent points (the tenant ran several top-ups), or if the auto-refund call to Stripe fails (Stripe temporarily down, the original card was closed by the issuing bank, etc.), the closure ledger marks the refund as 'operator-pending'. The wallet is still zeroed — the platform records the amount owed — and an Investatech operator reconciles manually via the Stripe Dashboard. You get an email confirming the closure either way; the multi-PI / failed-auto cases just take a few extra days to settle.

Subscription tier and Store availability

The Store is available to every tenant on every subscription tier — Basic, Premium, comp-Premium, trial, trial-expired tenants all see the sidebar entry and can top up + purchase as long as they have a connected Stripe-side payment method. There is no Premium gate, no per-product gate beyond what the platform team explicitly configures on a product. Some products may have their own restrictions (for example, a Premium-only training product is at the platform team's discretion to publish), but the module itself imposes no tier-based access. A trial-expired tenant can still top up + still spend wallet points — the Store is the one platform surface that explicitly remains usable after a Premium trial expires, because the wallet is yours and you should not lose access to the points you paid for.

Interactions with other modules

The Store sits underneath three other modules as the shared paid-action layer. The Fax module debits the Store wallet on every successful delivery (1 point per 10 pages, rounded up); the Store activity log's Faxes tab is the canonical place to see what every fax has cost. The Service Agreements module debits the wallet when you run an AI Review (1 point per run); the activity log surfaces those as inline-action-credit charges. The Investatech-tenant booking pages are downstream of book_meeting product purchases — buying the product seeds a pre-paid booking slot on the Investatech operator's calendar, and the booking lifecycle (confirmation email, calendar invite, reminders) is exactly the same as any tenant-to-tenant booking. The wallet's two-bucket model and the nonrefundable-first allocation flow uniformly across all three module integrations; you do not need to track which module debited the wallet on which row — the activity log surfaces everything in one place with a per-row source.

The Store wallet CANNOT be used to pay your Premium subscription. Premium is billed through a separate Stripe subscription on a different cadence, and the wallet is deliberately walled off from that flow — the design choice is that subscription billing should be a stable predictable recurring charge, not subject to the wallet being unexpectedly drained by usage. Likewise, the Store wallet cannot be transferred between tenants, cannot be

cashed out (the only path to dollar value is via the closure-refund flow), and cannot be assigned to another team member as a personal allowance. The wallet is a tenant-level resource.

Troubleshooting

- Wallet balance did not update after a top-up — Stripe's webhook is usually instant but can occasionally be delayed by a Vercel cold start or a Stripe Dashboard re-routing. The receipt page shows 'Payment is processing' until the webhook lands; refresh the page after 10-15 seconds. If the balance is still wrong after a minute, check the Activity panel — the top-up row appears the moment the row is created, even before the webhook finalises; if the row exists but the wallet has not bumped, contact support.
- Buy button returns 'insufficient points' but the wallet looks like it has enough — the buy route re-checks the wallet at submit time against the product's price. A race condition (another team member just spent points at the same moment) can cause the check to fail. Refresh the page so the wallet card reads the current balance; if you actually still have enough, click Buy again.
- Cancel button missing on a book_meeting order — once a meeting has been scheduled on the Investatech calendar, the order leaves the platform's self-serve cancel window; the Activity panel hides the Cancel button. Email the Investatech operator directly to discuss cancellation or rescheduling.
- Team member sees the Store sidebar entry but no buy buttons — the per-member can_store_purchase permission is off. Ask the Owner to flip it on at Settings ’ Team ’ that member ’ Module Access ’ 'Can make in-store purchases'. Server-side enforcement means the buy buttons hide for members without the permission, which is by design.
- Top-up button missing or disabled — only the Owner can fire a top-up. Other team members see the wallet card but no 'Add points' card. If the Owner is logged in and the button is still disabled, check that the Owner's

Stripe-side billing method is set (Settings ’ Billing); the top-up checkout needs a valid billing address for automatic tax calculation.

- Refund did not land on the original card after account closure — single-PI refunds typically land within 3-10 business days. Multi-PI tenants land on operator-pending review and take a few extra business days while an Investatech operator reconciles via the Stripe Dashboard. The closure email confirms whether your case is auto-refund or operator-pending; check that email first, then contact support with the closure date and the tenant slug if 14 business days have passed without the refund landing.
- Comp grant from the platform team did not appear — operator-issued grants land instantly on the activity log. If you were told a grant was issued and you do not see it, ask the operator for the grant reason and timestamp; the activity log shows every grant with its reason visible so the absence of a row means the grant did not actually fire.