vsmsconnect — API Reference

Backend HTTP API for the VSMS Connect fiscalisation platform.

Generated 2026-06-25 · 33 sections

Contents

vsmsconnect — API

137 endpoints across 31 groups.

businesses auth users invites audit api-keys locations certificates mcp health well-known invoices fiscalisation-jobs fiscal-records sync-from-date unmapped-tax-types auto-fiscalise xero xero-tax-mappings xero-webhook sage sage-tax-mappings sage200-tax-mappings myob myob-tax-mappings accounting-sync accounting-push accounting-push-tax-mappings print print-agent http-connector types

Authentication

Each endpoint's auth requirement is shown on its row. Response envelopes and payloads are defined under types.

businesses (6)

POST `/api/v1/businesses/register`

First-user business registration with TaxCore verification. Validates the supplied PFX/PAC/UID against V-SDC, creates the business, promotes the calling user to Administrator, and returns fresh JWT tokens. No role guard — the caller has no businessId yet.

POST `/api/v1/businesses`

Create an additional business. Used after the first business is registered. TIN must be unique across all businesses.

GET `/api/v1/businesses`

Returns the single business the authenticated user belongs to. Users without a businessId receive an empty list. Returned as an APIResponse list shape for FE consistency.

PATCH `/api/v1/businesses/:businessId`

Partially update non-credential business settings — name, address fields, currencyCode, and the demo Xero tenant id used for Training routing. At least one field must be provided.

PATCH /api/v1/businesses/:businessId/taxcore-config

Update V-SDC credentials for a business. multipart/form-data — every field optional, at least one required. When `pac` is supplied, the server verifies it against V-SDC `/status` (mTLS) before persisting. When `businessUID` is also supplied, `status.uid` is cross-checked against it.

POST /api/v1/businesses/:businessId/default-certificate

Registration wizard step 2 shortcut. Creates a Certificate row against the business's default Location and mirrors the same fields into the legacy `Businesses.Vsdc*` columns so the consumer's fallback path keeps working.

auth (9)

POST /api/v1/auth/login

Validate credentials and issue a JWT access token plus an httpOnly refresh-token cookie. Returns the user and businessId (null if onboarding incomplete). Constant-time dummy hash verify on unknown emails to prevent timing-based enumeration.

POST /api/v1/auth/register

Create the first user account. Hashes the password with argon2, creates a Users row with roles=[] and BusinessId=NULL, and fires AUTH_REGISTER + fire-and-forget welcome email. No tokens are issued — the client is expected to redirect to login.

POST /api/v1/auth/refresh

Rotate the refresh token. Reads the httpOnly refreshToken cookie (or body fallback for native clients), verifies RS256 signature, checks typ='refresh', re-fetches the user to detect deactivation and post-iat password changes, and issues a fresh access+refresh pair.

POST /api/v1/auth/logout

Clear the refresh-token cookie. Idempotent — returns 200 even when the cookie is absent or malformed. Decodes (does not verify) the cookie to write an AUTH_LOGOUT audit event with the userId/businessId when present.

GET /api/v1/auth/setup-status

Returns whether any user has been registered yet — used by the frontend to gate the registration form vs. invite-only flow.

POST /api/v1/auth/register-invited

Complete registration for a user who received an invite link. Validates the invite token in Redis (invite:{token}), creates the user with the invite's roles and organizationId, deletes the token (single-use), and fires AUTH_REGISTER + welcome email.

POST /api/v1/auth/change-password

Verify the caller's current password, hash the new one, and issue a fresh access+refresh token pair. Sets Users.PasswordChangedAt to invalidate all existing refresh tokens. Rate-limited via 'default' bucket.

POST /api/v1/auth/forgot-password

Request a 6-digit OTP reset code. Always returns 200 with a generic message — silently no-ops for unknown emails (anti-enumeration). Generates a code, SHA-256-hashes it into PasswordResetTokens, and XADDs the raw code to the auth:password-reset Redis Stream for the email worker. Rate-limited via 'default' bucket.

POST /api/v1/auth/reset-password

Confirm a password reset. Validates the OTP against the stored SHA-256 hash, checks expiry, deletes the token (single-use), hashes the new password with argon2, and sets PasswordChangedAt to invalidate every existing refresh token. All failure modes return the same RESET_TOKEN_INVALID code. Rate-limited via 'default' bucket.

users (3)

GET /api/v1/users

List all users belonging to the authenticated caller's business (businessId taken from the JWT, not the URL).

POST /api/v1/users

Create a new user inside the caller's business. Argon2-hashes the password, derives username from the email local part, inserts into Users + UserRoles, and fires USER_CREATED + welcome email (fire-and-forget).

PATCH /api/v1/users/:userId

Partial update of a user inside the caller's business. Patch may include roles, isActive, firstName, lastName, email, and/or password. Admin-removal safeguards prevent removing your own Administrator role or removing the last administrator. Fires USER_DEACTIVATED on isActive=false, USER_UPDATED otherwise.

invites (2)

POST /api/v1/invites

Create a one-time invite. Generates a UUID token, stores `invite:{uuid}` in Redis with TTL INVITE_TTL_SECONDS (default 72h) holding { email, roles, organizationId=businessId, invitedBy=actorUserId }, and sends an invite email. On email failure the Redis token is deleted to avoid orphans.

GET /api/v1/invites/:token

Validate an invite token and return its metadata so the registration page can pre-fill the email and show the role(s). Public — invite recipients have no account yet.

audit (3)

GET /api/v1/businesses/:businessId/audit-events

Paginated audit log for a business. Supports filters by eventType (comma-separated → IN clause), resourceType, resourceId, dateFrom/dateTo (epoch ms), and a search term (matches ResourceId or ActorEmail with %term%). Returns items + total + stats (fiscalised/failed/actors counts). Fires VIEW_AUDIT_LOG fire-and-forget after the response is sent.

GET /api/v1/businesses/:businessId/audit-events/export

Stream all matching audit events as CSV (Content-Disposition: attachment; filename=audit-log-<businessId>-<yyyy-mm-dd>.csv). Same filters as the list endpoint; Administrator only (requireAdmin). Registered before /:id so Express does not treat 'export' as an id param.

GET /api/v1/businesses/:businessId/audit-events/:id

Fetch a single audit log entry by its BIGINT AuditLogId. Scoped to the business.

api-keys (3)

POST /api/v1/businesses/:businessId/api-keys

Generate a new static API key for on-premise agents (Sage 100/300/200 Evolution, Amicus). Stores SHA-256(rawKey) + an 8-char prefix; returns the raw 64-hex-char key exactly once. Optional scope binds the key to one AccountingProviderType; optional locationId scopes it to one Location (validated against the business). Fires API_KEY_CREATED.

GET /api/v1/businesses/:businessId/api-keys

List all API keys for a business (active + revoked). Raw secret is never returned — only label, prefix, scope, locationId, createdAt, lastUsedAt, revokedAt.

DELETE /api/v1/businesses/:businessId/api-keys/:keyId

Soft-revoke an API key (sets RevokedAt). Idempotent — re-calling on an already-revoked key is a no-op. Fires API_KEY_REVOKED on first revocation.

locations (5)

GET /api/v1/businesses/:businessId/locations List all Locations for a business.

POST /api/v1/businesses/:businessId/locations

Create a new Location for the business. locationType defaults server-side; isDefault optionally promotes this Location to the business's default (existing default cleared atomically).

GET /api/v1/businesses/:businessId/locations/:locationId

Fetch a single Location scoped to the business.

PATCH /api/v1/businesses/:businessId/locations/:locationId

Partial update of a Location. isDefault=true is routed through the dedicated setDefault path so the previous default is cleared atomically. status='inactive' soft-disables the Location.

DELETE /api/v1/businesses/:businessId/locations/:locationId

Hard-delete a Location (or soft-delete depending on repository impl — see ILocationRepository.remove).

certificates (6)

GET /api/v1/businesses/:businessId/locations/:locationId/certificates

List all Certificates registered against a Location. Returns the PUBLIC projection — VSDC credentials (PFX filename, password, PAC) are stripped; `hasCertificate: boolean` indicates cert presence.

POST /api/v1/businesses/:businessId/locations/:locationId/certificates

Upload a new V-SDC certificate for a Location. multipart/form-data: PFX file + password + PAC + UID. The server RC2→3DES-converts the PFX, verifies the PAC against V-SDC /status, cross-checks UID against status.uid, writes the file under CERT_STORAGE_DIR/{uuid}.pfx, and persists a Certificates row. PFX bytes are cleaned up if persistence fails. PFX file max 5 MB.

GET

/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId

Fetch a single Certificate (public projection — credential fields stripped).

PATCH

/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId

Patch a Certificate's metadata — name, status (active|revoked), and/or isDefault. isDefault=true is routed through setLocationDefault for an atomic swap of the Location's default certificate.

POST

/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId/

Revoke a Certificate (sets Status='revoked'). Convenience action — equivalent to PATCH with status='revoked'.

DELETE

/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId

Delete a Certificate. Removes the DB row and unlinks the PFX file from CERT_STORAGE_DIR (fire-and-forget; unlink errors swallowed).

mcp (5)

GET /api/v1/mcp/connection

Return the public MCP server URL, transport, and tool list. Does NOT require an MCP token — only a valid JWT. Used by the app to display connection details before the user creates their first token.

POST /api/v1/mcp/tokens

Issue a 90-day RS256 personal-access token (typ='mcp') for AI agents. jti = McpTokens.McpTokenId UUID. Only SHA-256(rawToken) is persisted; the raw token is returned ONCE on creation and never again.

GET /api/v1/mcp/tokens

List the caller's active MCP tokens (non-revoked, non-expired). Raw token hash is never exposed — only id, name, createdAt, lastUsedAt, expiresAt.

DELETE /api/v1/mcp/tokens/:tokenId

Soft-revoke an MCP token (sets RevokedAt). Idempotent — re-calling on an already-revoked token is a no-op.

GET /api/v1/mcp/agent-config

Return a Claude Desktop / mcp.json configuration snippet referencing the MCP server URL and a placeholder Authorization header. The raw token is not returned — the user must substitute the value they saved at creation time.

health (1)

GET /health

Liveness probe. Returns `{ ok: true }` for a bare call. When `?services` is present (any value), the response also includes `services: { redis: boolean }` reflecting `Redis.testConnection()`. Unversioned (does NOT live under /api/v1).

well-known (2)

GET /.well-known/apple-app-site-association

iOS Universal Links association file. Returns a JSON payload listing the app's TeamID + bundleIdentifier under `applinks.details` with path scope `/go*`. Served unauthenticated and OUTSIDE the /api/v1 prefix — the OS fetches it directly over HTTPS. TeamID is a deploy-time placeholder.

GET /.well-known/assetlinks.json

Android App Links assetlinks. Returns a JSON array delegating common.handle_all_urls to the app's android_app namespace with package_name + sha256_cert_fingerprints. Served unauthenticated and OUTSIDE the /api/v1 prefix. Package name and fingerprint are deploy-time placeholders.

invoices (11)

GET /api/v1/businesses/:businessId/invoices/batches

Paginated list of import batches for the business. Each batch row carries row counts (imported/failed) and the originating file metadata. Registered before /:invoiceId so the literal 'batches' segment never collides with the dynamic id.

GET /api/v1/businesses/:businessId/invoices/daily-fiscal-report

Server-side aggregation of fiscalised payments inside a date window. Payment-aware — every fiscalised InvoicePayments row counts as one fiscal event and amounts come from PaymentAmount, so a multi-payment invoice with payments on different days splits correctly. Fires DAILY_FISCAL_REPORT_GENERATED (fire-and-forget) for audit traceability.

GET /api/v1/businesses/:businessId/invoices

Paginated invoice list for the business. RTK infinite-query compatible. Returns invoices in the same shape as the single-invoice GET (with embedded payments[] and line-items omitted).

GET /api/v1/businesses/:businessId/invoices/:invoiceId

Fetch a single invoice with its line items and payments[] embedded. Cross-tenant probing returns 404 since the repo filters by businessId.

GET /api/v1/businesses/:businessId/invoices/:invoiceId/payments

List every payment under a single invoice. Returns the same payments[] array embedded on GET /invoices/:invoiceId plus per-payment fiscal data (FiscalInvoiceNumber, FiscalTimestamp, etc.). Ordered by PaymentDate ASC, CreatedAt ASC. Lets the FE refresh only the payments list after a per-payment action without re-fetching the parent invoice.

GET /api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId

Direct drill-down for a single InvoicePayments row. Returns the same InvoicePaymentDTO shape that appears inside the parent invoice's payments[] array. Returns 404 when the payment doesn't belong to the (invoiceId, businessId) pair — protects against cross-tenant or cross-invoice probing.

POST /api/v1/businesses/:businessId/invoices/reprocess-unmapped

Re-evaluates every invoice blocked for MISSING_TAX_MAPPING against the current active TaxRateMappings. For each invoice whose codes are now fully mapped: updates line-item TaxLabel/TaxRatePercent in place, strips MISSING_TAX_MAPPING from FiscalisationBlockReasons, and sets EligibleForFiscalisation = 1. Idempotent — already-eligible invoices are untouched.

POST /api/v1/businesses/:businessId/invoices/cancel-bulk

Bulk cancel up to MAX_PAYMENTS_PER_FISCALISATION_JOB fiscalised payments in one job. Builds a cancellation row per valid payment, then dispatches every freshly-inserted cancellation in a single FiscalisationJob so the client polls one jobId. Invalid IDs come back in blocked[]; IDs whose original already has a cancellation in flight come back in inFlight[]. Fires INVOICE_PAYMENT_CANCELLATION_REQUESTED audit per dispatched payment.

POST

/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/cancel

Cancel a single fiscalised payment. Issues a cancellation row (Sale → Refund / Refund → Sale) with the same line items but BuyerTin set to the seller's TIN and ReferentDocumentNumber pointing back at the original SDC fiscal number, then dispatches it through the standard fiscalisation pipeline. Poll the returned jobId — on success the original payment is marked cancelled by the consumer. Fires INVOICE_PAYMENT_CANCELLATION_REQUESTED audit.

POST

/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/copy

Per-payment Copy. Inserts a new Invoices row with InvoiceType='COPY', CopiesInvoiceId pointing at the source invoice, totals scaled to the source payment's amount, plus a synthetic InvoicePayments row carrying CopiesPaymentId. Source payment's SDC FiscalInvoiceNumber is the V-SDC referent on the new submission; TransactionType is preserved (Copy of Sale → Sale, Copy of Refund → Refund). One-time per source payment (retry of a failed copy allowed). Immediately dispatches for fiscalisation and fires INVOICE_COPY_REQUESTED audit.

POST

/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/refund

Per-payment Refund. Inserts a new Invoices row with TransactionType='REFUND' and InvoiceType inherited from the source (Normal → Normal Refund, Advance → Advance Refund), RefundsInvoiceId pointing at the source invoice, totals scaled to the source payment's amount, plus a synthetic InvoicePayments row carrying RefundsPaymentId. Source payment's SDC FiscalInvoiceNumber is the V-SDC referent on the new submission. Only valid on a fiscalised SALE payment under a Normal or Advance invoice — refunding a refund is rejected. One-time per source payment (retry of a failed refund allowed). FE button is hidden on Normal/Advance (only surfaced for PROFORMA quotes); backend stays permissive so the service-layer REFUNDABLE_TYPES guard remains the source of truth. Fires INVOICE_REFUND_REQUESTED audit.

fiscalisation-jobs (3)

POST /api/v1/businesses/:businessId/fiscalisation-jobs

Trigger fiscalisation for a batch of payments. Pre-flight eligibility validator re-reads persisted EligibleForFiscalisation / FiscalisationBlockReasons + checks the business's TaxCore config, then partitions the batch into dispatched and blocked. Blocked payments come back with per-payment reason codes so operators can fix the root cause and retry. Batch size capped at MAX_PAYMENTS_PER_FISCALISATION_JOB (currently 10).

POST /api/v1/businesses/:businessId/fiscalisation-jobs/by-invoice

Invoice-level fiscalisation trigger for PROFORMA (quote) rows. Accepts invoiceIds[]; the service translates each to a synthetic payment (idempotent) before delegating to the standard per-payment dispatch. Same partitioned-response contract as POST /fiscalisation-jobs.

GET /api/v1/businesses/:businessId/fiscalisation-jobs/:jobId

Poll fiscalisation job status and per-payment results. The Results array carries per-payment fiscal data (FiscalInvoiceNumber, FiscalTimestamp, FiscalVerificationUrl, errorMessage, attempts). IDOR-guarded — a job belonging to another business returns 404. Add ?lite=true to omit the Results array — returns counts + status only, reducing repeated-poll payload from MB → ~200 bytes for large jobs.

fiscal-records (1)

GET /api/v1/businesses/:businessId/fiscal-records/:requestId

Fetches a previously fiscalised invoice directly from the TaxCore V-SDC by its 8-character RequestId (stored in InvoicePayments.FiscalRequestId after fiscalisation). Proxies to TaxCore GET /api/v3/invoices/{requestId} using mTLS (cert + pfxPassword + PAC). The encryptedInternalData field is stripped server-side before the response is returned to the client.

sync-from-date (1)

PATCH /api/v1/accounting/sync-from-date

Per-business sync lower-bound date — Administrator-only. The stored epoch-ms lower bound is read by all sync paths (manual Sync Now + scheduled workers for Xero/Sage/MYOB) and passed as modifiedSince to the accounting provider. Setting to null restores the default 'full history' behaviour. Webhooks are never filtered. Surfaced on GET /accounting/{xero|sage|myob}/status so the app can render the date picker pre-populated.

unmapped-tax-types (1)

GET /api/v1/businesses/:businessId/accounting/unmapped-tax-types

List provider tax codes that have arrived on invoices without a confirmed mapping. Populated fire-and-forget by sync services and webhook workers on every invoice ingestion pass. Each row includes providerTaxName and providerRate (nullable). Rows that now have an active TaxRateMappings entry are excluded server-side via NOT EXISTS join — confirming a mapping silently removes the entry; deactivating a mapping makes it reappear. Used by the home-screen banner and the Integrations nav badge to notify admins.

auto-fiscalise (1)

PATCH /api/v1/accounting/auto-fiscalise

Toggle the per-business Businesses.AutoFiscaliseInvoices flag (DB default 1). When on, every newly-inserted eligible invoice from any sync path (manual Sync Now, webhook, scheduled poll) is dispatched to TaxCore automatically. Fires AUTO_FISCALISE_ENABLED / AUTO_FISCALISE_DISABLED audit. Reflected on GET /accounting/{provider}/status.autoFiscaliseInvoices so the app can render the switch alongside connection state. Manual fiscalisation via POST /fiscalisation-jobs remains available regardless.

xero (6)

GET /api/v1/accounting/xero/start

Begin a Xero OAuth2 authorisation flow for the caller's business. Generates a PKCE verifier + state, persists them in OAuthStates, and returns the Xero authorisation URL the app/browser must redirect to. Rate-limited (10/min open bucket).

GET /api/v1/accounting/xero/callback

OAuth callback target hit by the user's browser after Xero consent. Validates the state (constant-time), exchanges the authorisation code, upserts XeroConnections rows for every tenant returned, and either 302-redirects (desktop) or serves a 200 text/html bridge page (mobile UA) that opens the native `vsms://auth/xero/...` deep link.

GET /api/v1/accounting/xero/status

Connection summary for the caller's business — whether a Xero connection exists, the active tenant, the list of authorised tenants (no tokens), and the current `autoFiscaliseInvoices` flag.

GET /api/v1/accounting/xero/tenants

List every Xero tenant authorised for the caller's business. Each entry carries `isActive` to indicate which tenant is the current target of sync / webhooks.

POST /api/v1/accounting/xero/tenants/:xeroTenantId/activate

Switch the active Xero tenant for the caller's business. Flips `IsActive` on XeroConnections rows so subsequent sync / webhook traffic uses the chosen tenant.

DELETE /api/v1/accounting/xero/connection

Hard-disconnect Xero for the caller's business — deletes every XeroConnections row, drops encrypted tokens, fires `XERO_DISCONNECTED` audit.

xero-tax-mappings (5)

GET /api/v1/accounting/xero/tax-mappings

List all Xero tax-type to V-SDC tax-label mappings for the business (active and inactive). Active rows ordered first, then alphabetically. `isActive: boolean` per row so the admin UI can surface a Restore action on soft-deleted entries.

POST /api/v1/accounting/xero/tax-mappings/auto-map

Fetch Xero `/TaxRates` from the active tenant + V-SDC `/status`, then return match suggestions with `needsReview` flags + alternatives. Does NOT persist — the admin confirms each suggestion via PUT. Rate-limited (10/min open bucket).

PUT /api/v1/accounting/xero/tax-mappings

Create or update one Xero tax-type to V-SDC label mapping (MERGE on (businessId, 'xero', providerTaxType)). Re-activates an existing inactive row instead of inserting a duplicate. Fires `XERO_TAX_MAPPING_UPDATED` audit.

DELETE /api/v1/accounting/xero/tax-mappings/:providerTaxType

Soft-delete a Xero tax mapping — sets `IsActive=0`. Row is retained for history and can be re-activated via the restore endpoint. Idempotent.

PATCH /api/v1/accounting/xero/tax-mappings/:providerTaxType/restore

Re-activate a previously soft-deleted Xero tax mapping (`IsActive=0` → `1`). Idempotent.

xero-webhook (1)

POST /webhooks/xero

Xero-initiated webhook delivery. Verifies HMAC-SHA256 over the raw request body using `XERO_WEBHOOK_SIGNING_KEY` (constant-time compared to the base64 `x-xero-signature` header). Valid signature → fan each event to the `xero:webhook:events` Redis Stream and respond `200` within Xero's 5-second ack budget. Xero's intent-to-receive handshake (`events: []`) flows through the same code path. NOT mounted under `/api/v1`.

sage (4)

GET /api/v1/accounting/sage/start

Begin a Sage OAuth2 authorisation flow for the caller's business. Generates PKCE + state, persists in OAuthStates, returns the Sage authorisation URL. Rate-limited (10/min open bucket).

GET /api/v1/accounting/sage/callback

Sage OAuth callback. Validates state, exchanges the code, upserts a single SageConnections row (Sage has one business per token — no tenant selection), then 302 redirects (desktop) or serves a 200 text/html bridge page (mobile) opening `vsms://auth/sage/{success|failure}`.

GET /api/v1/accounting/sage/status

Sage connection summary — whether a Sage connection exists, the connected Sage business, and the current `autoFiscaliseInvoices` flag.

DELETE /api/v1/accounting/sage/connection

Soft-deactivate the Sage connection for the caller's business (`IsActive=0`). Rows are retained for history; the user may re-authorise via `/sage/start`. Fires `SAGE_DISCONNECTED` audit.

sage-tax-mappings (5)

GET /api/v1/accounting/sage/tax-mappings

List all Sage tax-type to V-SDC tax-label mappings for the business (`Provider='sage'`). Mirrors `/accounting/xero/tax-mappings`.

POST /api/v1/accounting/sage/tax-mappings/auto-map

Pull Sage tax-rates + V-SDC `/status` and return match suggestions. Does NOT persist. Rate-limited (10/min open bucket).

PUT /api/v1/accounting/sage/tax-mappings

Create or update one Sage tax-mapping. Re-activates an inactive row on match. Fires `SAGE_TAX_MAPPING_UPDATED` audit (resource type `SAGE_CONNECTION`).

DELETE /api/v1/accounting/sage/tax-mappings/:providerTaxType

Soft-delete a Sage tax-mapping (`IsActive=0`). Idempotent.

PATCH /api/v1/accounting/sage/tax-mappings/:providerTaxType/restore

Re-activate a deactivated Sage tax-mapping (`IsActive=1`). Idempotent.

sage200-tax-mappings (5)

GET /api/v1/accounting/sage200evolution/tax-mappings

List all Sage 200 Evolution tax-mappings for the business (`Provider='sage200evolution'`). The path-param `:id` everywhere else in this surface is the Sage `idTaxRate` (the provider tax type).

POST /api/v1/accounting/sage200evolution/tax-mappings/auto

One-off auto-map against the customer's on-premise Sage 200 Evolution database. Opens a direct MSSQL connection using the supplied credentials, runs `SELECT idTaxRate, Code, cFiscalTaxLabel FROM dbo.TaxRate`, and upserts a proposed mapping for every row not already mapped. Distinct from the `auto-map` pattern used by Xero/Sage/MYOB — this endpoint reads the customer's own DB and persists proposals immediately.

PUT /api/v1/accounting/sage200evolution/tax-mappings/:id

Upsert a single Sage 200 Evolution mapping. `:id` is the Sage `idTaxRate` (provider tax type). Body carries the V-SDC label + optional metadata.

DELETE /api/v1/accounting/sage200evolution/tax-mappings/:id

Soft-delete a Sage 200 Evolution mapping (`IsActive=0`). Idempotent.

POST /api/v1/accounting/sage200evolution/tax-mappings/:id/restore

Re-activate a deactivated Sage 200 Evolution mapping (`IsActive=1`). Note: uses POST (unlike the Xero/Sage/MYOB restore endpoints which use PATCH). Idempotent.

myob (6)

GET /api/v1/accounting/myob/start

Begin a MYOB AccountRight Live OAuth2 authorisation flow. Generates PKCE + state, returns the MYOB authorisation URL. Rate-limited (10/min open bucket).

GET /api/v1/accounting/myob/callback

MYOB OAuth callback. Validates state, exchanges the code, upserts a MyobConnections row per discovered company file, then 302-redirects (desktop) or serves a 200 text/html bridge page (mobile) opening `vsms://auth/myob/{success|failure}`.

GET /api/v1/accounting/myob/status

MYOB connection summary — whether a connection exists, the file list, and the current `autoFiscaliseInvoices` flag.

GET /api/v1/accounting/myob/files

List every MYOB company file authorised for the caller's business, each carrying `isActive` indicating the current sync target.

POST /api/v1/accounting/myob/files/:myobFileId/activate

Switch the active MYOB company file for the caller's business. Fires `MYOB_FILE_ACTIVATED` audit.

DELETE /api/v1/accounting/myob/connection

Soft-deactivate all MYOB connections for the caller's business (`IsActive=0`). Rows are retained for history. Fires `MYOB_DISCONNECTED` audit.

myob-tax-mappings (5)

GET /api/v1/accounting/myob/tax-mappings

List all MYOB tax-mappings for the business (`Provider='myob'`). Mirrors `/accounting/sage/tax-mappings`.

POST /api/v1/accounting/myob/tax-mappings/auto-map

Pull MYOB tax codes + V-SDC `/status` and return match suggestions. Does NOT persist. Rate-limited (10/min open bucket).

PUT /api/v1/accounting/myob/tax-mappings

Create or update one MYOB tax-mapping. Fires `MYOB_TAX_MAPPING_UPDATED` audit.

DELETE /api/v1/accounting/myob/tax-mappings/:providerTaxType

Soft-delete a MYOB tax-mapping (`IsActive=0`). Idempotent.

PATCH /api/v1/accounting/myob/tax-mappings/:providerTaxType/restore

Re-activate a deactivated MYOB tax-mapping (`IsActive=1`). Idempotent.

accounting-sync (3)

GET /api/v1/accounting/xero/sync

Pull-sync PAID invoices from the active Xero tenant: fetch via `XeroProvider.getInvoices`, normalise to `AccountingInvoice[]`, persist to `Invoices` table (`Source='xero'`) with one `ImportBatches` row per call. Idempotency key = `SHA-256(businessId:xeroTenantId:xeroInvoiceId)`. When `AutoFiscaliseInvoices=1`, newly-inserted eligible invoices are chunked (10) and dispatched as fiscalisation jobs. Returns the `AccountingInvoice[]` plus a summary message including `jobIds=`. HTTP caching disabled.

GET /api/v1/accounting/sage/sync

Pull-sync invoices from the active Sage connection. Fetches OUTSTANDING / OVERDUE / PAID via three API calls merged client-side, normalises, persists to `Invoices` (`Source='sage'`). Idempotency key = `SHA-256(businessId:sageBusinessId:sageInvoiceId)`. `modifiedSince` is silently ignored (Sage's `sales_invoices` has no date filter). Auto-fiscalisation behaviour matches Xero. HTTP caching disabled.

GET /api/v1/accounting/myob/sync

Pull-sync `Closed` invoices from the active MYOB company file. Fetches `/Sale/Invoice/Service` and `/Sale/Invoice/Item` merged by UID, normalises, persists to `Invoices` (`Source='myob'`). Idempotency key = `SHA-256(businessId:myobFileId:myobInvoiceId)`. Auto-fiscalisation behaviour matches Xero. HTTP caching disabled.

accounting-push (5)

POST /api/v1/businesses/:businessId/accounting/sage100/sync

On-premise Sage 100 agent push. Headless agent POSTs a batch of normalised `AccountingInvoice[]` for the business; the helper inserts → dedupes → auto-fiscalises (when the flag is on). API key must have `scope=sage100` (enforced by `requireScope`); empty array is allowed (used by the agent's first-run dry-run wizard).

POST /api/v1/businesses/:businessId/accounting/sage300/sync

On-premise Sage 300 agent push. Same shape as the Sage 100 endpoint but scope-bound to `sage300`.

POST /api/v1/businesses/:businessId/accounting/sage200evolution/sync

On-premise Sage 200 Evolution agent push. Scope-bound to `sage200evolution`. Per-row register→till routing via `Tills.PosRegisterId`; unrouted invoices return `outcomes[].outcome='register_unmapped'` and fire `SAGE200_RECEIPT_UNROUTED` audit. Supports advance-sale deposits (one `AccountingInvoicePayment` per `_etblInvoiceDeposits` row, fan-out to N `NormalizedInvoicePayment`s).

POST /api/v1/businesses/:businessId/accounting/amicus/sync

On-premise AMICUS agent push. Scope-bound to `amicus`. Per-row register→till routing (null till → sign-but-don't-print), `PosSaleId` persisted on `Invoices`, refund referent resolution via `originalPosSaleId`. Supports split-tender (one payment with `tenders[]`). Unrouted invoices return `outcomes[].outcome='register_unmapped'` and fire `AMICUS_RECEIPT_UNROUTED` audit.

GET /api/v1/businesses/:businessId/accounting/sync/status

Return the last-push timestamp per on-premise source (sage100, sage300, sage200evolution, amicus) for the business. Reads `ImportBatches.CreatedAt` per source via `findLatestCreatedAtBySources`; null when the source has never pushed.

accounting-push-tax-mappings (13)

GET /api/v1/accounting/sage100/tax-mappings

List Sage 100 tax-mappings for the business (`Provider='sage100'`).

PUT /api/v1/accounting/sage100/tax-mappings

Upsert one Sage 100 tax-mapping. Same body shape as the other tax-mapping CRUD surfaces.

DELETE /api/v1/accounting/sage100/tax-mappings/:providerTaxType

Soft-delete a Sage 100 tax-mapping (`IsActive=0`). Idempotent.

PATCH /api/v1/accounting/sage100/tax-mappings/:providerTaxType/restore

Re-activate a deactivated Sage 100 mapping (`IsActive=1`). Idempotent.

GET /api/v1/accounting/sage300/tax-mappings

List Sage 300 tax-mappings for the business (`Provider='sage300'`).

PUT /api/v1/accounting/sage300/tax-mappings Upsert one Sage 300 tax-mapping.

DELETE /api/v1/accounting/sage300/tax-mappings/:providerTaxType

Soft-delete a Sage 300 tax-mapping (`IsActive=0`). Idempotent.

PATCH /api/v1/accounting/sage300/tax-mappings/:providerTaxType/restore

Re-activate a deactivated Sage 300 mapping (`IsActive=1`). Idempotent.

GET /api/v1/accounting/amicus/tax-mappings

List AMICUS tax-mappings for the business (`Provider='amicus'`).

PUT /api/v1/accounting/amicus/tax-mappings Upsert one AMICUS tax-mapping.

DELETE /api/v1/accounting/amicus/tax-mappings/:providerTaxType

Soft-delete an AMICUS tax-mapping (`IsActive=0`). Idempotent.

PATCH /api/v1/accounting/amicus/tax-mappings/:providerTaxType/restore

Re-activate a deactivated AMICUS mapping (`IsActive=1`). Idempotent.

POST /api/v1/accounting/amicus/tax-mappings/auto-seed

Pre-populate the fixed AMICUS taxonomy (`VAT` + `EXEMPT`) so the admin can confirm both V-SDC labels before any sale arrives. No body — the taxonomy is known by contract with the bundled agent profile, not discovered from the customer DB. AMICUS-only; no equivalent for Sage 100/300.

print (7)

GET /api/v1/businesses/:businessId/print/tills

List every registered till for the business — name, machine id, `IsDefault`, `IsOnline`, `LastSeenAt`, and optional `LocationId`.

PUT /api/v1/businesses/:businessId/print/tills/:tillId/name

Rename a till. Fires `TILL_RENAMED` audit (resource type `TILL`).

POST /api/v1/businesses/:businessId/print/tills/:tillId/test

Dispatch a test PrintJob (`type='TEST'`) to the till. The agent picks it up on its next poll or via the SSE wake stream. Fires `TEST_PRINT_SENT` audit.

POST /api/v1/businesses/:businessId/print/tills/setup-token

Generate a one-time setup token used by the print agent's first-run wizard to claim a permanent TillToken. Optionally binds the resulting till to a specific Location (TAXCORE-246) — single-location merchants omit and the claim resolves to the business default. Fires `TILL_SETUP_TOKEN_GENERATED` audit.

PATCH /api/v1/businesses/:businessId/print/default-till

Set the default till for the business. Future auto-print jobs that don't carry an explicit till routing key target this till. Fires `TILL_DEFAULT_SET` audit.

GET /api/v1/businesses/:businessId/print/jobs

List print jobs for the business, optionally filtered by status.

POST /api/v1/businesses/:businessId/print/jobs/:id/retry

Reset a `FAILED` or `STALE` print job back to `PENDING` so the agent will pick it up again. Fires `PRINT_JOB_RETRIED` audit.

print-agent (5)

POST /api/v1/print/agent/claim

Print-agent first-run setup. Exchanges the one-time SetupToken (issued via `POST /print/tills/setup-token`) for a permanent TillToken — the SetupToken itself IS the credential for this single call. Returns the till + plaintext token (only time it is returned). Fires `TILL_CLAIMED` audit.

GET /api/v1/print/agent/jobs

Return PENDING print jobs for the authenticated till. Polled by the agent as a fallback to the SSE wake channel.

GET /api/v1/print/agent/jobs/stream

Long-lived SSE channel — emits a `wake` event whenever a new PrintJob is published for this till so the agent can fetch + print without waiting for the next poll tick. Keep-alive `:keepalive` comment every 25 s to beat reverse-proxy idle timeouts.

POST /api/v1/print/agent/jobs/:id/ack

Agent ack — mark a print job `PRINTED` (success) or `FAILED` (USB / printer error). The repository asserts the job belongs to the authenticated till and is still in a settable state.

POST /api/v1/print/agent/heartbeat

Periodic heartbeat — updates `Tills.IsOnline=1` and `LastSeenAt=now`. Sent by the agent on every poll tick / SSE reconnect.

http-connector (4)

POST /api/v1/businesses/:businessId/fiscalise

Main ingestion endpoint. Persists the invoice + dispatches it to V-SDC and sync-waits up to ~10 s for the fiscal response. Returns 200 with `fiscalInvoiceNumber` + QR/URL/hash when the consumer reaches a terminal state inside the window; 202 with a `statusUrl` when the timeout elapses with payments still in-flight; 201 with a `triggerUrl` for PROFORMA bodies (caller must follow up with the trigger endpoint).

GET /api/v1/businesses/:businessId/fiscalise/:invoiceId

Idempotent status retrieval. Returns the same response shape as `POST /fiscalise` so a caller polling after a 202 gets a drop-in replacement once the consumer reaches a terminal state.

POST /api/v1/businesses/:businessId/fiscalise/:invoiceId/trigger

Explicit dispatch for a PROFORMA imported via POST /fiscalise's 201 path. Delegates to the existing `triggerFiscalisationByInvoiceIds` primitive — non-PROFORMA invoices are rejected with 422 INVOICE_NOT_QUOTE.

POST /api/v1/businesses/:businessId/fiscalise/cancel

Submit a V-SDC cancellation document for a previously-fiscalised payment. Resolves `fiscalInvoiceNumber` via `findByFiscalInvoiceNumber`, delegates to the per-payment `cancelPayment` primitive (sibling InvoicePayments row with `CancelsPaymentId`), and waits for the cancellation receipt to be signed before responding — so the caller gets back the full `fiscalJournal` + QR + signed hash in one round-trip and can print the cancellation receipt without polling. Falls through to 202 + `statusUrl` if the sync-wait window elapses.

vsmsconnect — API — accounting-push-tax-mappings

← all endpoints · types ↗

GET `/api/v1/accounting/sage100/tax-mappings` auth: jwt (administrator)

List Sage 100 tax-mappings for the business (`Provider='sage100'`).

handlers: AccountingAgentTaxMappingsController.listFor('sage100')

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

Sage 100 mappings.

PUT `/api/v1/accounting/sage100/tax-mappings` auth: jwt (administrator)

Upsert one Sage 100 tax-mapping. Same body shape as the other tax-mapping CRUD surfaces.

handlers: AccountingAgentTaxMappingsController.upsertFor('sage100')

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	Sage 100 tax-code.
`providerTaxName` body string | null	no	—	Human-readable tax name.
`providerRate` body number | null	no	—	Provider tax rate.
`vsdcTaxLabel` body string	yes	—	V-SDC label.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

Invalid V-SDC label.

DELETE `/api/v1/accounting/sage100/tax-mappings/:providerTaxType`

auth: jwt (administrator)

Soft-delete a Sage 100 tax-mapping (`IsActive=0`). Idempotent.

handlers: AccountingAgentTaxMappingsController.removeFor('sage100')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage 100 tax-code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH `/api/v1/accounting/sage100/tax-mappings/:providerTaxType/restore`

auth: jwt (administrator)

Re-activate a deactivated Sage 100 mapping (`IsActive=1`). Idempotent.

handlers: AccountingAgentTaxMappingsController.restoreFor('sage100')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage 100 tax-code to re-activate.

OUTPUTS

204

`empty`

No content.

GET `/api/v1/accounting/sage300/tax-mappings` auth: jwt (administrator)

List Sage 300 tax-mappings for the business (`Provider='sage300'`).

handlers: AccountingAgentTaxMappingsController.listFor('sage300')

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

Sage 300 mappings.

PUT `/api/v1/accounting/sage300/tax-mappings` auth: jwt (administrator)

Upsert one Sage 300 tax-mapping.

handlers: AccountingAgentTaxMappingsController.upsertFor('sage300')

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	Sage 300 tax-code.
`providerTaxName` body string | null	no	—	Human-readable tax name.
`providerRate` body number | null	no	—	Provider tax rate.
`vsdcTaxLabel` body string	yes	—	V-SDC label.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

Invalid V-SDC label.

DELETE `/api/v1/accounting/sage300/tax-mappings/:providerTaxType`

auth: jwt (administrator)

Soft-delete a Sage 300 tax-mapping (`IsActive=0`). Idempotent.

handlers: AccountingAgentTaxMappingsController.removeFor('sage300')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage 300 tax-code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH `/api/v1/accounting/sage300/tax-mappings/:providerTaxType/restore`

auth: jwt (administrator)

Re-activate a deactivated Sage 300 mapping (`IsActive=1`). Idempotent.

handlers: AccountingAgentTaxMappingsController.restoreFor('sage300')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage 300 tax-code to re-activate.

OUTPUTS

204

`empty`

No content.

GET `/api/v1/accounting/amicus/tax-mappings` auth: jwt (administrator)

List AMICUS tax-mappings for the business (`Provider='amicus'`).

handlers: AccountingAgentTaxMappingsController.listFor('amicus')

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

AMICUS mappings.

PUT `/api/v1/accounting/amicus/tax-mappings` auth: jwt (administrator)

Upsert one AMICUS tax-mapping.

handlers: AccountingAgentTaxMappingsController.upsertFor('amicus')

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	AMICUS tax-code (typically `VAT` or `EXEMPT`).
`providerTaxName` body string | null	no	—	Human-readable tax name.
`providerRate` body number | null	no	—	Provider tax rate.
`vsdcTaxLabel` body string	yes	—	V-SDC label.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

Invalid V-SDC label.

DELETE /api/v1/accounting/amicus/tax-mappings/:providerTaxType

auth: jwt (administrator)

Soft-delete an AMICUS tax-mapping (`IsActive=0`). Idempotent.

handlers: AccountingAgentTaxMappingsController.removeFor('amicus')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	AMICUS tax-code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH /api/v1/accounting/amicus/tax-mappings/:providerTaxType/restore

auth: jwt (administrator)

Re-activate a deactivated AMICUS mapping (`IsActive=1`). Idempotent.

handlers: AccountingAgentTaxMappingsController.restoreFor('amicus')

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	AMICUS tax-code to re-activate.

OUTPUTS

204

`empty`

No content.

POST `/api/v1/accounting/amicus/tax-mappings/auto-seed` auth: jwt (administrator)

Pre-populate the fixed AMICUS taxonomy (`VAT` + `EXEMPT`) so the admin can confirm both V-SDC labels before any sale arrives. No body — the taxonomy is known by contract with the bundled agent profile, not discovered from the customer DB. AMICUS-only; no equivalent for Sage 100/300.

handlers: AccountingAgentTaxMappingsController.autoSeedAmicus

INPUTS

No parameters.

OUTPUTS

> **200**
>
> APIResponseDataList<TaxRateMappingDTO>
>
> The two seeded mappings (VAT, EXEMPT).

vsmsconnect — API — accounting-push

← all endpoints · types ↗

POST `/api/v1/businesses/:businessId/accounting/sage100/sync` auth: apikey

On-premise Sage 100 agent push. Headless agent POSTs a batch of normalised `AccountingInvoice[]` for the business; the helper inserts → dedupes → auto-fiscalises (when the flag is on). API key must have `scope=sage100` (enforced by `requireScope`); empty array is allowed (used by the agent's first-run dry-run wizard).

handlers: AccountingAgentSyncController.syncSage100

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Must match the API key's `BusinessId`.
`Authorization` header string	yes	—	`ApiKey <key>` — validated by `requireApiKey` against `ApiKeys.KeyHash`.
`invoices` body PushInvoice[]	yes	—	Array of pushed invoices. Each entry mirrors `AccountingInvoice` with optional `posSaleId`/`posRegisterId`/`originalPosSaleId`/`cashierId`/`paym` `tenders[]`. Empty array allowed.

OUTPUTS

200

APIResponseDataObject<PushInvoicesResponse>

`{ fetched, inserted, duplicates, jobIds[], outcomes[] }`.

401

APIError

Missing or invalid API key.

403

APIError

API key scope does not match `sage100`.

POST `/api/v1/businesses/:businessId/accounting/sage300/sync` auth: apikey

On-premise Sage 300 agent push. Same shape as the Sage 100 endpoint but scope-bound to `sage300`.

handlers: AccountingAgentSyncController.syncSage300

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`Authorization` header string	yes	—	`ApiKey <key>`.
`invoices` body PushInvoice[]	yes	—	Array of pushed invoices.

OUTPUTS

200

APIResponseDataObject<PushInvoicesResponse>

Push result summary.

401

APIError

Missing or invalid API key.

403

APIError

API key scope does not match `sage300`.

POST `/api/v1/businesses/:businessId/accounting/sage200evolution/sync` auth: apikey

On-premise Sage 200 Evolution agent push. Scope-bound to `sage200evolution`. Per-row register→till routing via `Tills.PosRegisterId`; unrouted invoices return `outcomes[].outcome='register_unmapped'` and fire `SAGE200_RECEIPT_UNROUTED` audit. Supports advance-sale deposits (one `AccountingInvoicePayment` per `_etblInvoiceDeposits` row, fan-out to N `NormalizedInvoicePayment`s).

handlers: AccountingAgentSyncController.syncSage200Evolution

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`Authorization` header string	yes	—	`ApiKey <key>`.
`invoices` body PushInvoice[]	yes	—	Array of pushed invoices. Deposit rows arrive as separate payments under the same invoice.

OUTPUTS

200

APIResponseDataObject<PushInvoicesResponse>

Push result summary including per-invoice `outcomes` for unrouted receipts.

401

APIError

Missing or invalid API key.

403

APIError

API key scope does not match `sage200evolution`.

POST `/api/v1/businesses/:businessId/accounting/amicus/sync` auth: apikey

On-premise AMICUS agent push. Scope-bound to `amicus`. Per-row register→till routing (null till → sign-but-don't-print), `PosSaleId` persisted on `Invoices`, refund referent resolution via `originalPosSaleId`. Supports split-tender (one payment with `tenders[]`). Unrouted invoices return `outcomes[].outcome='register_unmapped'` and fire `AMICUS_RECEIPT_UNROUTED` audit.

handlers: AccountingAgentSyncController.syncAmicus

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`Authorization` header string	yes	—	`ApiKey <key>`.
`invoices` body PushInvoice[]	yes	—	Array of pushed invoices. Each may carry `posSaleId`, `posRegisterId`, `originalPosSaleId`, `cashierId`, and split-tender `payments[].tenders[]`.

OUTPUTS

200

APIResponseDataObject<PushInvoicesResponse>

Push result summary including per-invoice `outcomes` for unrouted receipts.

401

APIError

Missing or invalid API key.

403

APIError

API key scope does not match `amicus`.

GET `/api/v1/businesses/:businessId/accounting/sync/status`

auth: jwt (administrator | view_invoices | submit_invoices)

Return the last-push timestamp per on-premise source (sage100, sage300, sage200evolution, amicus) for the business. Reads `ImportBatches.CreatedAt` per source via `findLatestCreatedAtBySources`; null when the source has never pushed.

handlers: AccountingAgentSyncController.status

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.

OUTPUTS

200

APIResponseDataObject<PushStatusResponse>

Map of source → `{ lastPushAt: string | null }` (ISO-8601 timestamp).

403

APIError

USER_FORBIDDEN — businessId on path does not match the caller's JWT.

vsmsconnect — API — accounting-sync

← all endpoints · types ↗

GET `/api/v1/accounting/xero/sync` auth: jwt (administrator | view_invoices | submit_invoices)

Pull-sync PAID invoices from the active Xero tenant: fetch via `XeroProvider.getInvoices`, normalise to `AccountingInvoice[]`, persist to `Invoices` table (`Source='xero'`) with one `ImportBatches` row per call. Idempotency key = `SHA-256(businessId:xeroTenantId:xeroInvoiceId)`. When `AutoFiscaliseInvoices=1`, newly-inserted eligible invoices are chunked (10) and dispatched as fiscalisation jobs. Returns the `AccountingInvoice[]` plus a summary message including `jobIds=`. HTTP caching disabled.

handlers: AccountingInvoicesController.xeroSync

INPUTS

name	required	default	description
`contactId` query string	no	—	Provider-native contact GUID — narrows the result to invoices for that contact.
`modifiedSince` query ISO-8601 datetime	no	—	Only invoices updated after this date are returned (passed as `If-Modified-Since` to Xero).
`dateTo` query ISO-8601 datetime	no	—	Post-fetch filter on `updatedAt` (Xero has no `dateTo` param).
`page` query integer	no	—	1-based page. Omit to auto-paginate.
`pageSize` query integer	no	—	1–100. Capped by the provider.

OUTPUTS

200

APIResponseDataList<AccountingInvoice>

Provider-agnostic invoice list plus a summary message `fetched=X inserted=Y duplicates=Z[ jobIds=<uuid>,...]`.

401

APIError

REAUTH_REQUIRED — Xero refresh token rejected.

409

APIError

ACCOUNTING_PROVIDER_NOT_CONFIGURED — no active Xero connection.

429

APIError

XERO_RATE_LIMITED — propagated from Xero with `Retry-After`.

GET `/api/v1/accounting/sage/sync` auth: jwt (administrator | view_invoices | submit_invoices)

Pull-sync invoices from the active Sage connection. Fetches OUTSTANDING / OVERDUE / PAID via three API calls merged client-side, normalises, persists to `Invoices` (`Source='sage'`). Idempotency key = `SHA-256(businessId:sageBusinessId:sageInvoiceId)`. `modifiedSince` is silently ignored (Sage's `sales_invoices` has no date filter). Auto-fiscalisation behaviour matches Xero. HTTP caching disabled.

handlers: AccountingInvoicesController.sageSync

INPUTS

name	required	default	description
`contactId` query string	no	—	Sage contact id — narrows the result to one contact.
`modifiedSince` query ISO-8601 datetime	no	—	Silently ignored for Sage.
`dateTo` query ISO-8601 datetime	no	—	Post-fetch filter on `updatedAt`.
`page` query integer	no	—	1-based page.
`pageSize` query integer	no	—	1–100.

OUTPUTS

200

APIResponseDataList<AccountingInvoice>

Provider-agnostic invoice list plus summary message.

401

APIError

REAUTH_REQUIRED — Sage refresh token rejected.

409

APIError

ACCOUNTING_PROVIDER_NOT_CONFIGURED.

429

APIError

SAGE_RATE_LIMITED.

GET `/api/v1/accounting/myob/sync` auth: jwt (administrator | view_invoices | submit_invoices)

Pull-sync `Closed` invoices from the active MYOB company file. Fetches `/Sale/Invoice/Service` and `/Sale/Invoice/Item` merged by UID, normalises, persists to `Invoices` (`Source='myob'`). Idempotency key = `SHA-256(businessId:myobFileId:myobInvoiceId)`. Auto-fiscalisation behaviour matches Xero. HTTP caching disabled.

handlers: AccountingInvoicesController.myobSync

INPUTS

name	required	default	description
`contactId` query string	no	—	MYOB contact UID — narrows the result.
`modifiedSince` query ISO-8601 datetime	no	—	Forwarded to MYOB OData `$filter`.
`dateTo` query ISO-8601 datetime	no	—	Post-fetch filter on `updatedAt`.
`page` query integer	no	—	1-based page.
`pageSize` query integer	no	—	1–100.

OUTPUTS

200

APIResponseDataList<AccountingInvoice>

Provider-agnostic invoice list plus summary message.

401

APIError

MYOB_REAUTH_REQUIRED.

409

APIError

ACCOUNTING_PROVIDER_NOT_CONFIGURED.

429

APIError

MYOB_RATE_LIMITED.

vsmsconnect — API — api-keys

← all endpoints · types ↗

POST `/api/v1/businesses/:businessId/api-keys` auth: jwt (administrator)

Generate a new static API key for on-premise agents (Sage 100/300/200 Evolution, Amicus). Stores SHA-256(rawKey) + an 8-char prefix; returns the raw 64-hex-char key exactly once. Optional scope binds the key to one AccountingProviderType; optional locationId scopes it to one Location (validated against the business). Fires API_KEY_CREATED.

handlers: ApiKeyController.create

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Administrator bypass via requireBusinessAccess.
`label` body string	yes	—	Operator-facing label (1..200 chars).
`scope` body AccountingProviderType	no	—	Bind to a single provider (sage100, sage300, sage200evolution, amicus, ...). Null/omitted = unscoped.
`locationId` body uuid	no	—	Multi-location (TAXCORE-246) — scope the key to one Location. Validated against the business; mismatch returns 422 LOCATION_NOT_FOUND.

OUTPUTS

201

APIResponseDataObject<CreateApiKeyResponse>

ApiKeyDTO plus the raw `key` (64 hex chars). The raw key is shown ONCE and never returned again.

403

APIError

USER_FORBIDDEN — not Administrator or businessId mismatch.

422

APIError

LOCATION_NOT_FOUND — locationId does not belong to this business. VALIDATION_ERROR — schema violation.

GET `/api/v1/businesses/:businessId/api-keys` auth: jwt (administrator)

List all API keys for a business (active + revoked). Raw secret is never returned — only label, prefix, scope, locationId, createdAt, lastUsedAt, revokedAt.

handlers: ApiKeyController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.

OUTPUTS

200

APIResponseDataList<ApiKeyDTO>

All API keys (active and revoked) for the business.

403

APIError

USER_FORBIDDEN — not Administrator or businessId mismatch.

DELETE `/api/v1/businesses/:businessId/api-keys/:keyId` auth: jwt (administrator)

Soft-revoke an API key (sets RevokedAt). Idempotent — re-calling on an already-revoked key is a no-op. Fires API_KEY_REVOKED on first revocation.

handlers: ApiKeyController.revoke

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`keyId` path uuid	yes	—	ApiKeys.ApiKeyId to revoke.

OUTPUTS

204

`void`

Key revoked (or already-revoked).

403

`APIError`

USER_FORBIDDEN — not Administrator or businessId mismatch.

404

`APIError`

API_KEY_NOT_FOUND — keyId not in this business.

vsmsconnect — API — audit

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/audit-events` auth: jwt (administrator | view_audit)

Paginated audit log for a business. Supports filters by eventType (comma-separated → IN clause), resourceType, resourceId, dateFrom/dateTo (epoch ms), and a search term (matches ResourceId or ActorEmail with %term%). Returns items + total + stats (fiscalised/failed/actors counts). Fires VIEW_AUDIT_LOG fire-and-forget after the response is sent.

handlers: AuditController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Caller's JWT businessId must match (Administrator bypasses via requireBusinessAccess).
`page` query int	no	`1`	1-based page number; clamped to >= 1.
`pageSize` query int	no	`20`	Page size, clamped 1..100.
`eventType` query string	no	—	AuditEventType, optionally comma-separated for IN().
`resourceType` query string	no	—	Filter by ResourceType (INVOICE, USER, XERO_CONNECTION, ...).
`resourceId` query string	no	—	Filter by exact ResourceId.
`dateFrom` query int (epoch ms)	no	—	Inclusive lower bound on CreatedAt.
`dateTo` query int (epoch ms)	no	—	Inclusive upper bound on CreatedAt.
`search` query string	no	—	Substring match against ResourceId or ActorEmail.

OUTPUTS

200

`APIResponseDataObject<{ items: AuditLogDTO[]; total: number; page: number; totalPages: number; stats: { fiscalised: number; failed: number; actors: number } }>`

Paginated items plus computed stats.

403

APIError

USER_FORBIDDEN — businessId mismatch (non-Administrator) or insufficient role.

GET /api/v1/businesses/:businessId/audit-events/export

auth: jwt (administrator)

Stream all matching audit events as CSV (Content-Disposition: attachment; filename=audit-log-<businessId>-<yyyy-mm-dd>.csv). Same filters as the list endpoint; Administrator only (requireAdmin). Registered before /:id so Express does not treat 'export' as an id param.

handlers: AuditController.export

INPUTS

name	required	default	description
businessId path uuid	yes	—	Target business UUID. Administrator bypasses requireBusinessAccess.
eventType query string	no	—	AuditEventType, optionally comma-separated.
resourceType query string	no	—	Filter by ResourceType.
resourceId query string	no	—	Filter by exact ResourceId.
dateFrom query int (epoch ms)	no	—	Inclusive lower bound on CreatedAt.
dateTo query int (epoch ms)	no	—	Inclusive upper bound on CreatedAt.
search query string	no	—	Substring match against ResourceId or ActorEmail.

OUTPUTS

200 text/csv

text/csv

CSV stream with columns Id,CreatedAt,EventType,ActorEmail,ActorRole,ResourceType,ResourceId,IpAddress,Payload.

403

APIError

USER_FORBIDDEN — caller is not Administrator.

GET `/api/v1/businesses/:businessId/audit-events/:id` auth: jwt (administrator | view_audit)

Fetch a single audit log entry by its BIGINT AuditLogId. Scoped to the business.

handlers: AuditController.getById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`id` path int (BIGINT)	yes	—	AuditLogs.AuditLogId — numeric, parsed via parseInt.

OUTPUTS

200

APIResponseDataObject<AuditLogDTO>

Single audit log entry.

400

APIError

Invalid audit event id (id not parseable as integer).

403

APIError

USER_FORBIDDEN — businessId mismatch or insufficient role.

404

APIError

Audit event not found.

vsmsconnect — API — auth

← all endpoints · types ↗

POST `/api/v1/auth/login` auth: none

Validate credentials and issue a JWT access token plus an httpOnly refresh-token cookie. Returns the user and businessId (null if onboarding incomplete). Constant-time dummy hash verify on unknown emails to prevent timing-based enumeration.

handlers: AuthController.login

INPUTS

name	required	default	description
`email` body string (email)	yes	—	Account email address.
`password` body string	yes	—	Plaintext password (min 1 char per Zod; argon2-verified against stored hash).

OUTPUTS

200

APIResponseDataObject<LoginResponseDTO>

{ accessToken, refreshToken, businessId, user }. Refresh token also set as httpOnly cookie 'refreshToken' (sameSite strict, secure in non-dev).

401

APIError

AUTH_INVALID_CREDENTIALS (unknown email or wrong password) | AUTH_ACCOUNT_DEACTIVATED (user found but IsActive=0).

POST `/api/v1/auth/register` auth: none

Create the first user account. Hashes the password with argon2, creates a Users row with roles=[] and BusinessId=NULL, and fires AUTH_REGISTER + fire-and-forget welcome email. No tokens are issued — the client is expected to redirect to login.

handlers: AuthController.register

INPUTS

name	required	default	description
`firstName` body string	yes	—	Given name (min 1 char).
`lastName` body string	yes	—	Family name (min 1 char).
`username` body string	yes	—	Display username (min 1 char).
`email` body string (email)	yes	—	Email address — must be globally unique.
`password` body string	yes	—	Plaintext password (min 8 chars).
`confirmPassword` body string	yes	—	Must equal password (Zod refine).

OUTPUTS

201

APIResponseDataObject<RegisterResponseDTO>

{ message: 'Account created successfully' }.

409

APIError

AUTH_EMAIL_TAKEN — an account with this email already exists.

422

APIError

VALIDATION_ERROR — schema violation or passwords do not match.

POST /api/v1/auth/refresh auth: none

Rotate the refresh token. Reads the httpOnly refreshToken cookie (or body fallback for native clients), verifies RS256 signature, checks typ='refresh', re-fetches the user to detect deactivation and post-iat password changes, and issues a fresh access+refresh pair.

handlers: AuthController.refresh

INPUTS

name	required	default	description
`refreshToken` body string	no	—	Fallback for native clients whose OS HTTP stack does not reliably persist cookies. Cookie is the primary source.
`refreshToken` header cookie	no	—	httpOnly refreshToken cookie set on login.

OUTPUTS

200

`APIResponseDataObject<RefreshResponseDTO>`

{ accessToken, refreshToken, businessId, user }. New refresh-token cookie is set; the old one is invalidated by rotation.

401

`APIError`

AUTH_UNAUTHORIZED — missing/malformed/expired/tampered token, wrong typ, deactivated user, or token iat predates a password change.

POST /api/v1/auth/logout auth: none

Clear the refresh-token cookie. Idempotent — returns 200 even when the cookie is absent or malformed. Decodes (does not verify) the cookie to write an AUTH_LOGOUT audit event with the userId/businessId when present.

handlers: AuthController.logout

INPUTS

No parameters.

OUTPUTS

200

`APIResponseDataObject<{ message: string }>`

{ message: 'Logged out successfully' }. Refresh-token cookie is cleared.

GET `/api/v1/auth/setup-status` auth: none

Returns whether any user has been registered yet — used by the frontend to gate the registration form vs. invite-only flow.

handlers: AuthController.setupStatus

INPUTS

No parameters.

OUTPUTS

> **200**
>
> APIResponseDataObject<SetupStatusResponse>
>
> { isFirstUser: boolean } — true when Users count is 0.

POST `/api/v1/auth/register-invited` auth: none

Complete registration for a user who received an invite link. Validates the invite token in Redis (invite:{token}), creates the user with the invite's roles and organizationId, deletes the token (single-use), and fires AUTH_REGISTER + welcome email.

handlers: AuthController.registerInvited

INPUTS

name	required	default	description
`inviteToken` body uuid	yes	—	Invite token from the invite email link.
`firstName` body string	yes	—	Given name (min 1 char).
`lastName` body string	yes	—	Family name (min 1 char).
`password` body string	yes	—	Plaintext password (min 8 chars).
`confirmPassword` body string	yes	—	Must equal password (Zod refine).

OUTPUTS

201

APIResponseDataObject<RegisterResponseDTO>

{ message: 'Account created successfully. Please log in.' }.

404

APIError

INVITE_INVALID — invite token absent or expired in Redis.

409

APIError

INVITE_ALREADY_USED — the email is already registered (token is also invalidated).

422

APIError

VALIDATION_ERROR — schema violation or passwords do not match.

POST `/api/v1/auth/change-password` auth: jwt

Verify the caller's current password, hash the new one, and issue a fresh access+refresh token pair. Sets Users.PasswordChangedAt to invalidate all existing refresh tokens. Rate-limited via 'default' bucket.

handlers: AuthController.changePassword

INPUTS

name	required	default	description
`currentPassword` body string	yes	—	Existing password (argon2-verified).
`newPassword` body string	yes	—	New password (min 8 chars; must differ from currentPassword).
`confirmPassword` body string	yes	—	Must equal newPassword (Zod refine).

OUTPUTS

200

APIResponseDataObject<ChangePasswordResponseDTO>

{ accessToken, refreshToken }. Refresh-token cookie also rotated.

401

APIError

AUTH_CURRENT_PASSWORD_INCORRECT — current password did not match. AUTH_UNAUTHORIZED — no/invalid bearer token.

404

APIError

USER_NOT_FOUND — authenticated userId resolved to no row (unlikely).

422

APIError

VALIDATION_ERROR — newPassword === currentPassword or passwords do not match.

POST `/api/v1/auth/forgot-password` auth: none

Request a 6-digit OTP reset code. Always returns 200 with a generic message — silently no-ops for unknown emails (anti-enumeration). Generates a code, SHA-256-hashes it into PasswordResetTokens, and XADDs the raw code to the auth:password-reset Redis Stream for the email worker. Rate-limited via 'default' bucket.

handlers: AuthController.forgotPassword

INPUTS

name	required	default	description
`email` body string (email)	yes	—	Email address. Normalised to lowercase trim before lookup.

OUTPUTS

> **200**
>
> APIResponseDataObject<ForgotPasswordResponseDTO>
>
> { message: 'If an account exists for this email, a reset code has been sent.' } — generic regardless of email validity.

> **422**
>
> APIError
>
> VALIDATION_ERROR — schema violation.

POST `/api/v1/auth/reset-password` auth: none

Confirm a password reset. Validates the OTP against the stored SHA-256 hash, checks expiry, deletes the token (single-use), hashes the new password with argon2, and sets PasswordChangedAt to invalidate every existing refresh token. All failure modes return the same RESET_TOKEN_INVALID code. Rate-limited via 'default' bucket.

handlers: AuthController.resetPassword

INPUTS

name	required	default	description
`email` body string (email)	yes	—	Email address.
`resetCode` body string	yes	—	6-digit OTP from the email.
`newPassword` body string	yes	—	New password (min 8 chars).
`confirmPassword` body string	yes	—	Must equal newPassword (Zod refine).

OUTPUTS

200

APIResponseDataObject<ResetPasswordResponseDTO>

{ message: 'Password reset successfully. Please sign in.' }.

400

APIError

RESET_TOKEN_INVALID — unknown email, no active token, expired, or wrong code (all collapsed under one code for anti-enumeration).

422

APIError

VALIDATION_ERROR — schema violation or passwords do not match.

vsmsconnect — API — auto-fiscalise

← all endpoints · types ↗

PATCH `/api/v1/accounting/auto-fiscalise` auth: jwt (administrator)

Toggle the per-business Businesses.AutoFiscaliseInvoices flag (DB default 1). When on, every newly-inserted eligible invoice from any sync path (manual Sync Now, webhook, scheduled poll) is dispatched to TaxCore automatically. Fires AUTO_FISCALISE_ENABLED / AUTO_FISCALISE_DISABLED audit. Reflected on GET /accounting/{provider}/status.autoFiscaliseInvoices so the app can render the switch alongside connection state. Manual fiscalisation via POST /fiscalisation-jobs remains available regardless.

handlers: AutoFiscaliseController.update

INPUTS

Request body type: AutoFiscaliseUpdateBody

name	required	default	description
enabled body boolean	yes	—	true → auto-fiscalise on incoming invoices; false → manual review only.

OUTPUTS

200

APIResponseDataObject<SetAutoFiscaliseResult>

Persisted result (typically { enabled: boolean }).

401

APIError

AUTH_UNAUTHORIZED — caller has no JWT user.

409

APIError

BUSINESS_CONTEXT_REQUIRED — no businessId on the JWT.

422

APIError

VALIDATION_ERROR — enabled not a boolean.

vsmsconnect — API — businesses

← all endpoints · types ↗

POST `/api/v1/businesses/register` auth: jwt

First-user business registration with TaxCore verification. Validates the supplied PFX/PAC/UID against V-SDC, creates the business, promotes the calling user to Administrator, and returns fresh JWT tokens. No role guard — the caller has no businessId yet.

handlers: BusinessesController.registerBusiness

INPUTS

name	required	default	description
`file` body multipart .pfx	yes	—	PFX certificate file (.pfx / .p12). Converted RC2 → 3DES on the server before storage.
`name` body string	yes	—	Business display name.
`tin` body string	yes	—	Tax Identification Number. Must be unique across all businesses.
`pac` body string	yes	—	PAC (Privileged Access Code) issued by TaxCore.
`uid` body string	yes	—	8-character uppercase alphanumeric UID assigned by TaxCore. Cross-checked against V-SDC `/status`.
`pfxPassword` body string	yes	—	Passphrase for the uploaded PFX certificate.
`street` body string | null	no	—	Optional street address.
`city` body string | null	no	—	Optional city.
`country` body string | null	no	—	ISO 3166-1 alpha-2 code (e.g. 'VU', 'AU').
`currencyCode` body string	no	VUV	ISO 4217 three-letter currency code. Defaults to 'VUV' when omitted.

OUTPUTS

> **201**
> APIResponseDTO<{ business: PublicBusinessDTO; accessToken: string; refreshToken: string }>

Business created. Sets a httpOnly refresh-token cookie. The `business` is the public projection (credential fields stripped).

409

APIError

BUSINESS_TIN_TAKEN — the supplied TIN already belongs to another business.

422

APIError

CERTIFICATE_INVALID — missing or unreadable PFX, or wrong passphrase. VSDC_PAC_INVALID / VSDC_UID_MISMATCH on V-SDC verification failure.

POST `/api/v1/businesses` auth: jwt (administrator)

Create an additional business. Used after the first business is registered. TIN must be unique across all businesses.

handlers: BusinessesController.create

INPUTS

Request body type: CreateBusinessRequest

name	required	default	description
`name` body string	yes	—	Business display name.
`tin` body string	yes	—	Tax Identification Number. Must be unique across all businesses.
`street` body string | null	no	—	Street address.
`city` body string | null	no	—	City.
`country` body string | null	no	—	ISO 3166-1 alpha-2 code (e.g. 'VU').
`currencyCode` body string	no	VUV	ISO 4217 three-letter currency code. Defaults to 'VUV' when omitted.

OUTPUTS

201

APIResponseDataObject<PublicBusinessDTO>

Newly created business. `vsdcPfxPassword`, `vsdcPac`, `vsdcCertificateFile` are stripped server-side.

409

APIError

BUSINESS_TIN_TAKEN.

422

APIError

VALIDATION_ERROR for schema violations.

GET `/api/v1/businesses` auth: jwt

Returns the single business the authenticated user belongs to. Users without a businessId receive an empty list. Returned as an APIResponse list shape for FE consistency.

handlers: BusinessesController.list

INPUTS

No parameters.

OUTPUTS

> **200**
>
> APIResponseDataList<PublicBusinessDTO>
>
> List with zero or one entry. Credential fields are stripped.

PATCH `/api/v1/businesses/:businessId` auth: jwt (administrator | update_business_settings)

Partially update non-credential business settings — name, address fields, currencyCode, and the demo Xero tenant id used for Training routing. At least one field must be provided.

handlers: BusinessesController.updateSettings

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Must match the caller's JWT businessId — Administrators bypass via requireBusinessAccess.
`name` body string	no	—	Updated display name (non-empty).
`street` body string | null	no	—	Pass null to clear.
`city` body string | null	no	—	Pass null to clear.
`country` body string | null	no	—	ISO 3166-1 alpha-2 code, or null to clear.
`currencyCode` body string	no	—	ISO 4217 three-letter currency code.
`demoXeroTenantId` body uuid | null	no	—	Xero tenant id that routes invoices through V-SDC Training mode. Empty string is normalised to null. Pass null to disable Training routing.

OUTPUTS

200

APIResponseDataObject<PublicBusinessDTO>

Updated business (credential fields stripped).

404

APIError

BUSINESS_NOT_FOUND.

422

APIError

VALIDATION_ERROR — empty body or schema violation.

PATCH `/api/v1/businesses/:businessId/taxcore-config`

auth: jwt (administrator | update_taxcore_credentials)

Update V-SDC credentials for a business. multipart/form-data — every field optional, at least one required. When `pac` is supplied, the server verifies it against V-SDC `/status` (mTLS) before persisting. When `businessUID` is also supplied, `status.uid` is cross-checked against it.

handlers: BusinessesController.updateTaxcoreConfig

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`file` body multipart .pfx	no	—	New PFX certificate. Converted RC2 → 3DES and written under CERT_STORAGE_DIR; only the UUID filename is stored in DB. Replaces any prior cert.
`password` body string | null	no	—	PFX passphrase (also stored as VsdcPfxPassword). Null clears the field. Write-only — never returned.
`pac` body string | null	no	—	PAC. Verified against V-SDC `/status` before being persisted. Null clears. Write-only.
`businessUID` body string | null	no	—	8-character uppercase alphanumeric UID. Cross-checked against status.uid when pac is also supplied.

OUTPUTS

200

`APIResponseDataObject<PublicBusinessDTO>`

Updated business. `hasCertificate: boolean` is included instead of the cert filename.

404

`APIError`

BUSINESS_NOT_FOUND.

422

`APIError`

VALIDATION_ERROR (no fields) | CERTIFICATE_INVALID (bad PFX / wrong passphrase / cert lacks the TaxCore.PublicConfiguration.TaxCoreApiUrl extension) | VSDC_PAC_INVALID | VSDC_UID_MISMATCH | VSDC_URL_NOT_FOUND_IN_CERT.

502

`APIError`

VSDC_UNAVAILABLE — network/server failure when verifying the PAC.

POST `/api/v1/businesses/:businessId/default-certificate`

auth: jwt (administrator | update_taxcore_credentials)

Registration wizard step 2 shortcut. Creates a Certificate row against the business's default Location and mirrors the same fields into the legacy `Businesses.Vsdc*` columns so the consumer's fallback path keeps working.

handlers: BusinessesController.createDefaultCertificate

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`file` body multipart .pfx	yes	—	PFX certificate file.
`password` body string	yes	—	PFX passphrase.
`pac` body string	yes	—	PAC issued by TaxCore.
`uid` body string	yes	—	8-character uppercase alphanumeric UID.
`name` body string	no	—	Optional certificate display name. Falls back server-side to the default Location's name.

OUTPUTS

201

APIResponseDataObject<PublicCertificateDTO>

Newly created certificate row (public projection). Legacy `Businesses.Vsdc*` columns are also updated server-side.

404

APIError

LOCATION_NOT_FOUND — the business has no default Location (legacy business missed the back-seed migration).

422

APIError

CERTIFICATE_INVALID — missing or unreadable PFX, or wrong passphrase.

vsmsconnect — API — certificates

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/locations/:locationId/certificates`

auth: jwt (administrator | update_taxcore_credentials)

List all Certificates registered against a Location. Returns the PUBLIC projection — VSDC credentials (PFX filename, password, PAC) are stripped; `hasCertificate: boolean` indicates cert presence.

handlers: CertificatesController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID.

OUTPUTS

200

APIResponseDataList<PublicCertificateDTO>

Certificates for the Location (credential fields stripped).

POST `/api/v1/businesses/:businessId/locations/:locationId/certificates`

auth: jwt (administrator | update_taxcore_credentials)

Upload a new V-SDC certificate for a Location. multipart/form-data: PFX file + password + PAC + UID. The server RC2→3DES-converts the PFX, verifies the PAC against V-SDC /status, cross-checks UID against status.uid, writes the file under CERT_STORAGE_DIR/{uuid}.pfx, and persists a Certificates row. PFX bytes are cleaned up if persistence fails. PFX file max 5 MB.

handlers: CertificatesController.create

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID — validated against the business.
`file` body multipart .pfx	yes	—	PFX certificate (max 5 MB). RC2-40-CBC accepted; normalised to 3DES on write.
`password` body string	yes	—	PFX passphrase. Stored AES-256-GCM encrypted.
`pac` body string	yes	—	PAC issued by TaxCore. Verified against V-SDC /status before persistence.
`uid` body string	yes	—	8-character uppercase alphanumeric UID. Cross-checked against status.uid.
`name` body string	no	—	Optional certificate display name. Falls back to the parent Location's name when blank.
`isDefault` body boolean | 'true' | '1'	no	—	Promote this certificate to the Location's default. Multipart string values 'true' / '1' are coerced to true.

OUTPUTS

201

APIResponseDataObject<PublicCertificateDTO>

Newly created Certificate (public projection).

404

APIError

LOCATION_NOT_FOUND — locationId does not belong to the business.

422

APIError

CERTIFICATE_INVALID — missing/unreadable PFX or wrong passphrase. VSDC_PAC_INVALID / VSDC_UID_MISMATCH / VSDC_URL_NOT_FOUND_IN_CERT on V-SDC verification failure. VALIDATION_ERROR for schema violations.

502

APIError

VSDC_UNAVAILABLE — network/server failure during PAC verification.

GET

`/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId`

auth: jwt (administrator | update_taxcore_credentials)

Fetch a single Certificate (public projection — credential fields stripped).

handlers: CertificatesController.getById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID.
`certificateId` path uuid	yes	—	Target Certificate UUID.

OUTPUTS

200

APIResponseDataObject<PublicCertificateDTO>

Certificate (public projection).

404

APIError

CERTIFICATE_NOT_FOUND.

PATCH

`/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId`

auth: jwt (administrator | update_taxcore_credentials)

Patch a Certificate's metadata — name, status (active|revoked), and/or isDefault. isDefault=true is routed through setLocationDefault for an atomic swap of the Location's default certificate.

handlers: CertificatesController.update

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID.
`certificateId` path uuid	yes	—	Target Certificate UUID.
`name` body string	no	—	Updated display name (min 1 char).
`status` body 'active' | 'revoked'	no	—	Revocation toggle.
`isDefault` body boolean	no	—	Promote to the Location's default certificate.

OUTPUTS

200

APIResponseDataObject<PublicCertificateDTO>

Updated Certificate (public projection).

404

APIError

CERTIFICATE_NOT_FOUND.

422

APIError

VALIDATION_ERROR — empty body or schema violation.

POST

`/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId/`

auth: jwt (administrator | update_taxcore_credentials)

Revoke a Certificate (sets Status='revoked'). Convenience action — equivalent to PATCH with status='revoked'.

handlers: CertificatesController.revoke

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID.
`certificateId` path uuid	yes	—	Target Certificate UUID.

OUTPUTS

200

APIResponseDataObject<PublicCertificateDTO>

Revoked Certificate (public projection).

404

APIError

CERTIFICATE_NOT_FOUND.

DELETE

`/api/v1/businesses/:businessId/locations/:locationId/certificates/:certificateId`

auth: jwt (administrator | update_taxcore_credentials)

Delete a Certificate. Removes the DB row and unlinks the PFX file from CERT_STORAGE_DIR (fire-and-forget; unlink errors swallowed).

handlers: CertificatesController.remove

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Parent Location UUID.
`certificateId` path uuid	yes	—	Target Certificate UUID.

OUTPUTS

204

`void`

Certificate removed.

404

APIError

CERTIFICATE_NOT_FOUND.

vsmsconnect — API — fiscal-records

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/fiscal-records/:requestId`

auth: jwt (administrator | view_audit)

Fetches a previously fiscalised invoice directly from the TaxCore V-SDC by its 8-character RequestId (stored in InvoicePayments.FiscalRequestId after fiscalisation). Proxies to TaxCore GET /api/v3/invoices/{requestId} using mTLS (cert + pfxPassword + PAC). The encryptedInternalData field is stripped server-side before the response is returned to the client.

handlers: FiscalRecordsController.getByRequestId

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`requestId` path string (^[A-Z0-9]{8}$)	yes	—	8-character uppercase alphanumeric V-SDC RequestId — e.g. 'JNSMBCXG'.

OUTPUTS

200

APIResponseDataObject<FiscalInvoiceRecordDTO>

Fiscal invoice record retrieved from V-SDC, with encryptedInternalData stripped.

404

APIError

BUSINESS_NOT_FOUND | FISCAL_RECORD_NOT_FOUND (V-SDC returned 404 for the RequestId).

422

APIError

CERTIFICATE_NOT_CONFIGURED (cert/PAC/password missing) | VSDC_URL_NOT_FOUND_IN_CERT (TaxCoreApiUrl not extracted from the cert).

502

APIError

FISCAL_ERROR — V-SDC returned an unexpected error.

vsmsconnect — API — fiscalisation-jobs

← all endpoints · types ↗

POST `/api/v1/businesses/:businessId/fiscalisation-jobs`

auth: jwt (administrator | submit_invoices)

Trigger fiscalisation for a batch of payments. Pre-flight eligibility validator re-reads persisted EligibleForFiscalisation / FiscalisationBlockReasons + checks the business's TaxCore config, then partitions the batch into dispatched and blocked. Blocked payments come back with per-payment reason codes so operators can fix the root cause and retry. Batch size capped at MAX_PAYMENTS_PER_FISCALISATION_JOB (currently 10).

handlers: FiscalisationJobsController.trigger

INPUTS

Request body type: TriggerFiscalisationBody

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`paymentIds` body uuid[]	yes	—	1..MAX_PAYMENTS_PER_FISCALISATIO payment UUIDs to dispatch.
`options.omitTextualRepresentation` body 0 | 1	no	`0`	0 = include textual representation (def 1 = omit.
`options.omitQRCodeGen` body 0 | 1	no	`0`	0 = generate QR code (default); 1 = sk
`tillId` body uuid | null	no	—	Optional till to print receipts to after fiscalisation completes.

OUTPUTS

200

`APIResponseDataObject<{ message: string; skippedIds: string[] }>`

outcome = 'already_fiscalised' — every payment was already terminal; nothing dispatched.

202

`APIResponseDataObject<{ jobId: string; skippedIds?: string[]; blocked?: TriggerBlockedPayment[]; warning?: string }>`

outcome = 'queued' — job dispatched. skippedIds/blocked/warning included when partial.

404

`APIError`

BUSINESS_NOT_FOUND.

422

> APIResponseDataObject<{ message: string; blocked: TriggerBlockedPayment[] }> | APIError
>
> outcome = 'all_blocked' — every payment failed eligibility (returned as a structured body, not a thrown APIError). Also: CERTIFICATE_NOT_CONFIGURED | VALIDATION_ERROR (schema bound exceeded).

POST `/api/v1/businesses/:businessId/fiscalisation-jobs/by-invoice`

auth: jwt (administrator | submit_invoices)

Invoice-level fiscalisation trigger for PROFORMA (quote) rows. Accepts invoiceIds[]; the service translates each to a synthetic payment (idempotent) before delegating to the standard per-payment dispatch. Same partitioned-response contract as POST /fiscalisation-jobs.

handlers: FiscalisationJobsController.triggerByInvoice

INPUTS

Request body type: TriggerFiscalisationByInvoiceBody

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceIds` body uuid[]	yes	—	1..MAX_PAYMENTS_PER_FISCALISATIO invoice UUIDs. Each is translated to a synthetic payment idempotently.
`options.omitTextualRepresentation` body 0 | 1	no	`0`	0 = include textual representation (def 1 = omit.
`options.omitQRCodeGen` body 0 | 1	no	`0`	0 = generate QR code (default); 1 = sk

OUTPUTS

200

APIResponseDataObject<{ message: string; skippedIds: string[] }>

outcome = 'already_fiscalised'.

202

APIResponseDataObject<{ jobId: string; skippedIds?: string[]; blocked?: TriggerBlockedPayment[]; warning?: string }>

outcome = 'queued'. Partial-success metadata included when present.

404

APIError

BUSINESS_NOT_FOUND | INVOICE_NOT_FOUND.

422

APIResponseDataObject<{ message: string; blocked: TriggerBlockedPayment[] }> | APIError

outcome = 'all_blocked'. Also: INVOICE_NOT_QUOTE | CERTIFICATE_NOT_CONFIGURED | VALIDATION_ERROR.

GET `/api/v1/businesses/:businessId/fiscalisation-jobs/:jobId`

auth: jwt (administrator | view_invoices | submit_invoices)

Poll fiscalisation job status and per-payment results. The Results array carries per-payment fiscal data (FiscalInvoiceNumber, FiscalTimestamp, FiscalVerificationUrl, errorMessage, attempts). IDOR-guarded — a job belonging to another business returns 404. Add ?lite=true to omit the Results array — returns counts + status only, reducing repeated-poll payload from MB → ~200 bytes for large jobs.

handlers: FiscalisationJobsController.getById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`jobId` path uuid	yes	—	FiscalisationJob UUID. Must belong to businessId.
`lite` query 'true' | 'false'	no	—	When 'true', omits the per-invoice Results array from the response. Useful for high-frequency polling. Schema (GetFiscalisationJobQuerySchema) is defined but the controller reads the raw query value directly — only the literal string 'true' triggers lite mode.

OUTPUTS

200

APIResponseDataObject<FiscalisationJobDTO>

Job record with results: [] when lite=true, otherwise full Results array.

404

APIError

FISCAL_JOB_NOT_FOUND — job does not exist or belongs to a different business (IDOR guard).

vsmsconnect — API — health

← all endpoints · types ↗

GET `/health` auth: none

Liveness probe. Returns `{ ok: true }` for a bare call. When `?services` is present (any value), the response also includes `services: { redis: boolean }` reflecting `Redis.testConnection()`. Unversioned (does NOT live under /api/v1).

handlers: Controllers.Health (platform/http/express/controllers/health.controller.ts)

INPUTS

name	required	default	description
`services` query string (presence only)	no	—	When present, the response includes a per-dependency boolean map. The value is ignored.

OUTPUTS

200

```
{ ok: boolean; services?: Record<string, boolean> }
```

Bare JSON object — NOT wrapped in APIResponse.

500

APIError

Unexpected error from the dependency probe.

vsmsconnect — API — http-connector

← all endpoints · types ↗

Getting started

The HTTP connector is a machine-to-machine API. Before your integration code can call it, a human admin at the customer's company must complete a one-time setup in our app to obtain three credentials.

Step 1 — Account + business onboarding (one-time, per customer)

The admin signs up at our app, verifies their email, and registers their business (uploads V-SDC certificate, validates PAC, enters TaxCore UID + licence key). This produces a unique `BUSINESS_ID` UUID tied to the business's fiscalisation credentials.

Step 2 — Generate an HTTP-scoped API key

In the app: **Integrations** → **Generic HTTP** → **Issue new key**. The admin picks a **Location** from the dropdown — the key is bound to that location's default V-SDC certificate and fiscal counter (one key per shop / outlet). A business with multiple locations issues one key per location. A one-shot reveal modal then displays three values, each with a copy button:

- **Backend URL** — e.g. `https://api.vsmsconnect.com/api/v1`
- **Business ID** — UUID, e.g. `7f3a1e2c-...`
- **API key** — 64-char hex string, e.g. `a3f2b1c8d9e7f6a4b2c1d8e9f7a6b5c4d3e2f1a0b9c8d7e6f5a4b3c2d1e0f9a8`

The API key is shown **once**. The customer's admin must copy it and hand it (plus the Backend URL and Business ID) to their integration team — usually via their internal password manager. If the key is lost, the admin revokes it and issues a new one. Backend URL and Business ID remain visible in the app's API Keys list.

Step 3 — Configure your integration

Set three environment variables (or config equivalents) in your code:

```
VSMS_CONNECT_BACKEND_URL  = "https://api.vsmsconnect.com/api/v1"
VSMS_CONNECT_BUSINESS_ID  = "7f3a1e2c-..."
VSMS_CONNECT_API_KEY      = "a3f2b1c8d9e7f6a4..."
```

Every request goes to:

```
POST {VSMS_CONNECT_BACKEND_URL}/businesses/{VSMS_CONNECT_BUSINESS_ID}/fiscalise
```

with headers:

```
Authorization: ApiKey {VSMS_CONNECT_API_KEY}
Idempotency-Key: <a uuid you generate per request>
Content-Type: application/json
```

Per-location keys

API keys are **location-scoped**. A business with multiple shops registers each shop as a `Location` in the app and issues a separate API key per location — the key binds every fiscalisation request to that location's default V-SDC certificate, fiscal counter, and receipt header. Multi-location customers therefore end up with one `Business ID` + N `API keys` (one per location). Your integration code typically routes the right `API key` per outlet via configuration; the `Backend URL` and `Business ID` are constant across the customer's whole estate.

Multi-tenant integrations

If your software integrates with multiple of our customers, each customer's admin completes Step 1 + Step 2 independently. You end up with one `Business ID` per customer plus one API key per (customer, location) pair. The Backend URL is the same for every customer.

Tax labels

Each line item carries a `taxLabel` — the V-SDC label, sent directly. No admin tax-mapping configuration is required for this connector (matches CSV import semantics). The valid label set:

label	meaning
A	Standard rate (Vanuatu VAT 15%)
B	Reserved (rate > 0, label B)
C	Reserved (rate > 0, label C)
D	Reserved (rate > 0, label D)
E	Reserved (rate > 0, label E)
F	Reserved (rate > 0, label F)
N	Zero-rated (rate = 0)

label	meaning
P	Exempt
T	Out of scope

Case matrix

One endpoint, `POST /fiscalise`, handles every fiscalisation case via two body fields: `invoiceType` (NORMAL / COPY / PROFORMA / ADVANCE) and `transactionType` (SALE / REFUND), plus an optional `training: true` flag that overrides `invoiceType` to `TRAINING`. The other three routes are operational helpers:

- `GET /fiscalise/:invoiceId` — idempotent status retrieval after a `202` from POST.
- `POST /fiscalise/:invoiceId/trigger` — explicit dispatch for a PROFORMA imported via the `201` path.
- `POST /fiscalise/cancel` — submit a V-SDC cancellation document by `fiscalInvoiceNumber`.

POST `/api/v1/businesses/:businessId/fiscalise` auth: apikey (scope=http)

Main ingestion endpoint. Persists the invoice + dispatches it to V-SDC and sync-waits up to ~10 s for the fiscal response. Returns 200 with `fiscalInvoiceNumber` + QR/URL/hash when the consumer reaches a terminal state inside the window; 202 with a `statusUrl` when the timeout elapses with payments still in-flight; 201 with a `triggerUrl` for PROFORMA bodies (caller must follow up with the trigger endpoint).

handlers: FiscaliseController.fiscalise

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID (must match the API key's BusinessId).
`Authorization` header string	yes	—	`ApiKey <raw key>` — key must be scoped to `ConnectorType.Http`.
`Idempotency-Key` header string	no	—	Stable UUID per request. A replay within ~5 min returns the cached response without re-running the handler.
`sync_timeout_ms` query number	no	`10000`	Sync-wait window in milliseconds. Capped at 30000.
`invoiceNumber` body string	yes	—	Caller's invoice number. Printed on the V-SDC receipt AND used as the dedup key — `SHA-256(businessId:http:invoiceNumber)` becomes the persisted `IdempotencyKey`. Re-sending the same value returns `409 INVOICE_DUPLICATE`.
`invoiceType` body 'NORMAL' | 'COPY' | 'PROFORMA' | 'ADVANCE'	yes	—	Case selection. TRAINING is a separate top-level flag — not an invoiceType value.
`transactionType` body 'SALE' | 'REFUND'	yes	—	Sale or refund.
`training` body boolean	no	—	When `true`, stamps `invoiceType = TRAINING` regardless of the caller-supplied value. Mutually exclusive with `invoiceType: 'COPY'` (422 `TRAINING_COPY_NOT_SUPPORTED`).
`source` body { referencedFiscalNumber: string; referencedFiscalTimestamp: string }	no	—	Required when `invoiceType=COPY`; forbidden otherwise. Carries the source payment's V-SDC fiscal number +

name	required	default	description
			timestamp (stored from a prior 200 response).
`referentDocumentNumber` body string	no	—	Required when `transactionType=REFUND` and the source was externally-fiscalised (paired with `referentDocumentDT`).
`referentDocumentDT` body string	no	—	ISO-8601 timestamp of the referent document. Pairs with `referentDocumentNumber`.
`reference` body string	no	—	PROFORMA → NORMAL conversion key. The fiscalisation module's `linkInvoiceToQuote` stamps `ConvertedFromQuoteId` and carries the proforma's SDC referent through automatically.
`invoiceDate` body number | string	yes	—	Epoch ms or ISO-8601. Required for non-COPY.
`currencyCode` body string	yes	—	ISO-4217 three-letter currency code. Required for non-COPY.
`cashierId` body string	yes	—	Cashier / operator id sent to V-SDC as the `cashier` field — printed on every fiscal receipt as the 'Cashier:' line. Required everywhere (Vanuatu V-SDC mandate); whitespace-only values are rejected with `VALIDATION_ERROR`.
`buyer` body { tin?: string; name?: string; costCentreId?: string; email?: string } | null	no	—	Optional. Presence triggers the 'with buyer' V-SDC wire body. V-SDC carries TIN + name + costCentreId on the wire receipt; `email` is stored locally and used by the auto-email pipeline to send the customer a copy of the fiscalised receipt when the per-connector auto-email toggle is enabled.
`lineItems` body Array<{ description, quantity, unitPrice, taxLabel, taxRatePercent, lineSubtotal, lineTaxAmount, lineTotal, gtin?, sortOrder?, itemAdditionalFields? }>	yes	—	Required non-empty for non-COPY. `taxLabel` must be one of the V-SDC labels (A/B/C/D/E/F/N/P/T).
`payments` body Array<{ amount, paymentType, paymentDate, externalPaymentId?, tenders? }>	yes	—	Required non-empty for non-COPY. For ADVANCE deposit chains: send ONE invoice with multiple payments — the consumer auto-chains ADVANCE → ADVANCE → … →

name	required	default	description
			NORMAL. Each payment may carry a `tenders[]` breakdown for split-tender.
`subtotalAmount` body number	yes	—	Sum of `lineItems[].lineSubtotal` (±0.01 reconciliation).
`taxAmount` body number	yes	—	Sum of `lineItems[].lineTaxAmount` (±0.01 reconciliation).
`totalAmount` body number	yes	—	Sum of `lineItems[].lineTotal` (±0.01 reconciliation). Must also equal sum of `payments[].amount`.
`invoiceAdditionalFields` body Record<string, string>	no	—	Optional bag forwarded verbatim to V-SDC as `invoiceAdditionalFields`.
`paymentAdditionalFields` body Record<string, string>	no	—	Optional bag forwarded verbatim to V-SDC inside the payment array.

OUTPUTS

200

APIResponseDataObject<FiscaliseSuccessResponse>

Every payment reached a terminal state inside the sync-wait window. `paymentResults[].fiscalInvoiceNumber` carries the SDC reference — store it for later refunds / copies / cancellations.

201

APIResponseDataObject<FiscaliseAcceptedResponse>

PROFORMA imported but not auto-dispatched. Body includes `triggerUrl`; call it via `POST /fiscalise/:invoiceId/trigger` to fiscalise.

202

APIResponseDataObject<FiscaliseSuccessResponse>

Timeout elapsed with payments still in-flight. Body includes `statusUrl`; poll `GET /fiscalise/:invoiceId` for the terminal state.

409

APIError

`INVOICE_DUPLICATE` — an invoice with this `invoiceNumber` already exists for this business.

422

APIError

`VALIDATION_ERROR` — schema or cross-field validation failed. Inspect `validationErrors[]` for per-field codes (`REFUND_REFERENT_REQUIRED`, `INVALID_COPY_BODY`, `TRAINING_COPY_NOT_SUPPORTED`, `LINE_SUM_MISMATCH`, `PAYMENT_SUM_MISMATCH`, `TENDER_SUM_MISMATCH`).

502

APIError

> `FISCAL_ERROR` — V-SDC rejected every payment. Inspect `GET /fiscalise/:invoiceId` for per-payment error detail.

GET `/api/v1/businesses/:businessId/fiscalise/:invoiceId` auth: apikey (scope=http)

Idempotent status retrieval. Returns the same response shape as `POST /fiscalise` so a caller polling after a 202 gets a drop-in replacement once the consumer reaches a terminal state.

handlers: FiscaliseController.getStatus

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Invoice id returned by a prior POST /fiscalise.
`Authorization` header string	yes	—	`ApiKey <raw key>` scoped to `ConnectorType.Http`.

OUTPUTS

> **200**
> APIResponseDataObject<FiscaliseSuccessResponse>
> Current per-payment snapshot.

> **404**
> APIError
> `INVOICE_NOT_FOUND` — invoice id is unknown under this business.

POST `/api/v1/businesses/:businessId/fiscalise/:invoiceId/trigger`

auth: apikey (scope=http)

Explicit dispatch for a PROFORMA imported via POST /fiscalise's 201 path. Delegates to the existing `triggerFiscalisationByInvoiceIds` primitive — non-PROFORMA invoices are rejected with 422 INVOICE_NOT_QUOTE.

handlers: FiscaliseController.trigger

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	PROFORMA invoice id.
`Authorization` header string	yes	—	`ApiKey <raw key>` scoped to `ConnectorType.Http`.
`Idempotency-Key` header string	no	—	Stable UUID per request.

OUTPUTS

202

`APIResponseDataObject<{ invoiceId; outcome; jobId; statusUrl }>`

Dispatched. Poll `statusUrl` for terminal state.

422

`APIError`

`INVOICE_NOT_QUOTE` — only PROFORMA invoices are eligible for this endpoint.

POST `/api/v1/businesses/:businessId/fiscalise/cancel` auth: apikey (scope=http)

Submit a V-SDC cancellation document for a previously-fiscalised payment. Resolves `fiscalInvoiceNumber` via `findByFiscalInvoiceNumber`, delegates to the per-payment `cancelPayment` primitive (sibling InvoicePayments row with `CancelsPaymentId`), and waits for the cancellation receipt to be signed before responding — so the caller gets back the full `fiscalJournal` + QR + signed hash in one round-trip and can print the cancellation receipt without polling. Falls through to 202 + `statusUrl` if the sync-wait window elapses.

handlers: FiscaliseController.cancel

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`Authorization` header string	yes	—	`ApiKey <raw key>` scoped to `ConnectorType.Http`.
`Idempotency-Key` header string	no	—	Stable UUID per request.
`fiscalInvoiceNumber` body string	yes	—	SDC `fiscalInvoiceNumber` returned by a prior 200 response.

OUTPUTS

200

APIResponseDataObject<FiscaliseCancelResponse>

Cancellation reached terminal state inside the sync-wait window. `paymentResults[0]` carries the cancellation receipt's own SDC `fiscalInvoiceNumber`, `fiscalJournal` (textual receipt), `fiscalQrCode`, `fiscalVerificationUrl`, `fiscalSignedHash`, and `fiscalTimestamp`. `cancellationPaymentId` echoes the new sibling payment row for correlation.

202

APIResponseDataObject<FiscaliseCancelResponse>

Timeout elapsed with the cancellation row still in-flight. `statusUrl` (`GET /fiscalise/:invoiceId`) returns the terminal cancellation once signed.

404

APIError

`INVOICE_NOT_FOUND` — no payment with that fiscal number exists under this business.

409

APIError

`INVOICE_CANCELLATION_IN_FLIGHT` — another cancellation for the same payment is already in flight.

422

APIError

> `INVOICE_NOT_CANCELLABLE` — payment is not fiscalised yet, missing fiscal reference, or wrong invoiceType. Body carries the specific code.

vsmsconnect — API — invites

← all endpoints · types ↗

POST `/api/v1/invites` auth: jwt (administrator)

Create a one-time invite. Generates a UUID token, stores `invite:{uuid}` in Redis with TTL INVITE_TTL_SECONDS (default 72h) holding { email, roles, organizationId=businessId, invitedBy=actorUserId }, and sends an invite email. On email failure the Redis token is deleted to avoid orphans.

handlers: InvitesController.create

INPUTS

name	required	default	description
`email` body string (email)	yes	—	Invitee email address.
`roles` body UserRole[]	yes	—	Roles to assign on registration (min 1).

OUTPUTS

201
APIResponseDataObject<InviteDTO>
{ inviteToken, email, roles, organizationId, inviteUrl } — inviteUrl is APP_URL/register-invited?token=<uuid>.

401
APIError
AUTH_UNAUTHORIZED — missing/invalid bearer token.

403
APIError
USER_FORBIDDEN — caller is not Administrator.

422
APIError
VALIDATION_ERROR — schema violation.

502
APIError
Bubbled email-service failure (token already cleaned up).

GET `/api/v1/invites/:token` auth: none

Validate an invite token and return its metadata so the registration page can pre-fill the email and show the role(s). Public — invite recipients have no account yet.

handlers: InvitesController.validate

INPUTS

name	required	default	description
`token` path uuid	yes	—	Invite token from the email link.

OUTPUTS

200

APIResponseDataObject<InviteDTO>

{ inviteToken, email, roles, organizationId }.

404

APIError

INVITE_INVALID — token missing from Redis (expired or never existed).

vsmsconnect — API — invoices

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/invoices/batches`

auth: jwt (administrator | view_invoices | submit_invoices)

Paginated list of import batches for the business. Each batch row carries row counts (imported/failed) and the originating file metadata. Registered before /:invoiceId so the literal 'batches' segment never collides with the dynamic id.

handlers: InvoicesController.listBatches

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Enforced by requireBusinessAccess (Administrator bypasses).
`offset` query integer	no	`0`	Pagination offset. Coerced from string; min 0.
`limit` query integer	no	`50`	Page size. Coerced from string; min 1, max 200.

OUTPUTS

200

APIResponseDataList<ImportBatchDTO>

Page of import batches with totalRows + hasMore.

GET `/api/v1/businesses/:businessId/invoices/daily-fiscal-report`

auth: jwt (administrator | view_audit | submit_invoices)

Server-side aggregation of fiscalised payments inside a date window. Payment-aware — every fiscalised InvoicePayments row counts as one fiscal event and amounts come from PaymentAmount, so a multi-payment invoice with payments on different days splits correctly. Fires DAILY_FISCAL_REPORT_GENERATED (fire-and-forget) for audit traceability.

handlers: InvoicesController.getDailyFiscalReport

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`dateFrom` query ISO-8601 datetime	yes	—	Inclusive start of the report window. Normalised to start-of-day server-side. Must be on or before dateTo (Zod refine).
`dateTo` query ISO-8601 datetime	yes	—	Inclusive end of the report window. Normalised to end-of-day (23:59:59.999 UTC) server-side.

OUTPUTS

200

APIResponseDataObject<DailyFiscalReportResponse>

Aggregated payment-level fiscal report for the window.

422

APIError

VALIDATION_ERROR — non-ISO date or dateFrom > dateTo.

GET `/api/v1/businesses/:businessId/invoices`

auth: jwt (administrator | view_invoices | submit_invoices)

Paginated invoice list for the business. RTK infinite-query compatible. Returns invoices in the same shape as the single-invoice GET (with embedded payments[] and line-items omitted).

handlers: InvoicesController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`offset` query integer	no	`0`	Pagination offset. min 0.
`limit` query integer	no	`50`	Page size. min 1, max 200.

OUTPUTS

200

APIResponseDataList<InvoiceDTO>

Page of invoices with totalRows + hasMore.

GET `/api/v1/businesses/:businessId/invoices/:invoiceId`

auth: jwt (administrator | view_invoices | submit_invoices)

Fetch a single invoice with its line items and payments[] embedded. Cross-tenant probing returns 404 since the repo filters by businessId.

handlers: InvoicesController.getById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Invoice UUID.

OUTPUTS

200

APIResponseDataObject<InvoiceDTO>

Invoice with line items and embedded payments[].

404

APIError

INVOICE_NOT_FOUND — invoice does not exist under this business.

GET `/api/v1/businesses/:businessId/invoices/:invoiceId/payments`

auth: jwt (administrator | view_invoices | submit_invoices)

List every payment under a single invoice. Returns the same payments[] array embedded on GET /invoices/:invoiceId plus per-payment fiscal data (FiscalInvoiceNumber, FiscalTimestamp, etc.). Ordered by PaymentDate ASC, CreatedAt ASC. Lets the FE refresh only the payments list after a per-payment action without re-fetching the parent invoice.

handlers: InvoicesController.listPayments

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Parent invoice UUID. Existence under businessId is verified before payments are listed (prevents cross-tenant probing of payment IDs).

OUTPUTS

> **200**
>
> APIResponseDataObject<GetInvoicePaymentsResponse>
>
> { payments: InvoicePaymentDTO[] } — possibly empty.

> **404**
>
> APIError
>
> INVOICE_NOT_FOUND — invoice does not exist under this business.

GET `/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId`

auth: jwt (administrator | view_invoices | submit_invoices)

Direct drill-down for a single InvoicePayments row. Returns the same InvoicePaymentDTO shape that appears inside the parent invoice's payments[] array. Returns 404 when the payment doesn't belong to the (invoiceId, businessId) pair — protects against cross-tenant or cross-invoice probing.

handlers: InvoicesController.getPaymentById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Parent invoice UUID.
`paymentId` path uuid	yes	—	Payment UUID. Must belong to invoiceId under businessId.

OUTPUTS

200

APIResponseDataObject<GetInvoicePaymentByIdResponse>

Single InvoicePaymentDTO.

404

APIError

INVOICE_NOT_FOUND — payment missing or under a different invoice/business.

POST `/api/v1/businesses/:businessId/invoices/reprocess-unmapped`

auth: jwt (administrator)

Re-evaluates every invoice blocked for MISSING_TAX_MAPPING against the current active TaxRateMappings. For each invoice whose codes are now fully mapped: updates line-item TaxLabel/TaxRatePercent in place, strips MISSING_TAX_MAPPING from FiscalisationBlockReasons, and sets EligibleForFiscalisation = 1. Idempotent — already-eligible invoices are untouched.

handlers: InvoicesController.reprocessUnmapped

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.

OUTPUTS

200

`APIResponseDataObject<{ reprocessed: number; nowEligible: number; stillBlocked: number }>`

Counts of invoices touched and how many became eligible.

POST `/api/v1/businesses/:businessId/invoices/cancel-bulk`

auth: jwt (administrator | submit_invoices)

Bulk cancel up to MAX_PAYMENTS_PER_FISCALISATION_JOB fiscalised payments in one job. Builds a cancellation row per valid payment, then dispatches every freshly-inserted cancellation in a single FiscalisationJob so the client polls one jobId. Invalid IDs come back in blocked[]; IDs whose original already has a cancellation in flight come back in inFlight[]. Fires INVOICE_PAYMENT_CANCELLATION_REQUESTED audit per dispatched payment.

handlers: InvoicesController.cancelBulk

INPUTS

Request body type: CancelInvoicesBulkBody

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`paymentIds` body uuid[]	yes	—	1..MAX_PAYMENTS_PER_FISCALISATION_JOB payment UUIDs to cancel. Validated by CancelInvoicesBulkBodySchema.

OUTPUTS

202

`APIResponseDataObject<BulkCancelInvoiceResponse>`

{ jobIds, cancelledPaymentIds, blocked[], inFlight[] } — jobIds is empty when nothing was dispatched.

422

`APIError`

INVOICE_CANCELLATION_BULK_EMPTY | CERTIFICATE_NOT_CONFIGURED | VALIDATION_ERROR (schema bound exceeded).

POST

`/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/cancel`

auth: jwt (administrator | submit_invoices)

Cancel a single fiscalised payment. Issues a cancellation row (Sale → Refund / Refund → Sale) with the same line items but BuyerTin set to the seller's TIN and ReferentDocumentNumber pointing back at the original SDC fiscal number, then dispatches it through the standard fiscalisation pipeline. Poll the returned jobId — on success the original payment is marked cancelled by the consumer. Fires INVOICE_PAYMENT_CANCELLATION_REQUESTED audit.

handlers: InvoicesController.cancel

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Parent invoice UUID.
`paymentId` path uuid	yes	—	Payment UUID to cancel. Must be fiscalised and not already in flight.

OUTPUTS

202

APIResponseDataObject<CancelInvoiceResponse>

{ jobId, cancellationPaymentId } — poll the job for terminal status.

409

APIError

INVOICE_CANCELLATION_IN_FLIGHT | INVOICE_NOT_CANCELLABLE | INVOICE_ALREADY_CANCELLED.

422

APIError

INVOICE_TYPE_NOT_SUPPORTED_FOR_CANCELLATION | CERTIFICATE_NOT_CONFIGURED.

POST

`/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/copy`

auth: jwt (administrator | submit_invoices)

Per-payment Copy. Inserts a new Invoices row with InvoiceType='COPY', CopiesInvoiceId pointing at the source invoice, totals scaled to the source payment's amount, plus a synthetic InvoicePayments row carrying CopiesPaymentId. Source payment's SDC FiscalInvoiceNumber is the V-SDC referent on the new submission; TransactionType is preserved (Copy of Sale → Sale, Copy of Refund → Refund). One-time per source payment (retry of a failed copy allowed). Immediately dispatches for fiscalisation and fires INVOICE_COPY_REQUESTED audit.

handlers: InvoicesController.copy

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Source invoice UUID.
`paymentId` path uuid	yes	—	Source payment UUID. Must be a fiscalised payment under a supported invoice type.

OUTPUTS

202

APIResponseDataObject<CopyInvoiceResponse>

{ invoiceId, invoicePaymentId, invoiceNumber, jobId } — jobId is null when triggerFiscalisation did not queue (e.g. already_fiscalised / all_blocked).

404

APIError

INVOICE_NOT_FOUND.

409

APIError

INVOICE_ALREADY_COPIED.

422

APIError

INVOICE_NOT_COPYABLE | INVOICE_TYPE_NOT_SUPPORTED_FOR_COPY | CERTIFICATE_NOT_CONFIGURED.

POST

`/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/refund`

auth: jwt (administrator | submit_invoices)

Per-payment Refund. Inserts a new Invoices row with TransactionType='REFUND' and InvoiceType inherited from the source (Normal → Normal Refund, Advance → Advance Refund), RefundsInvoiceId pointing at the source invoice, totals scaled to the source payment's amount, plus a synthetic InvoicePayments row carrying RefundsPaymentId. Source payment's SDC FiscalInvoiceNumber is the V-SDC referent on the new submission. Only valid on a fiscalised SALE payment under a Normal or Advance invoice — refunding a refund is rejected. One-time per source payment (retry of a failed refund allowed). FE button is hidden on Normal/Advance (only surfaced for PROFORMA quotes); backend stays permissive so the service-layer REFUNDABLE_TYPES guard remains the source of truth. Fires INVOICE_REFUND_REQUESTED audit.

handlers: InvoicesController.refund

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`invoiceId` path uuid	yes	—	Source invoice UUID.
`paymentId` path uuid	yes	—	Source SALE payment UUID. Must be fiscalised and not already refunded.

OUTPUTS

202

APIResponseDataObject<RefundPaymentResponse>

{ invoiceId, invoicePaymentId, invoiceNumber, jobId } — jobId null when not queued.

404

APIError

INVOICE_NOT_FOUND.

409

APIError

INVOICE_ALREADY_REFUNDED.

422

APIError

> INVOICE_NOT_REFUNDABLE | INVOICE_TYPE_NOT_SUPPORTED_FOR_REFUND | CERTIFICATE_NOT_CONFIGURED.

vsmsconnect — API — locations

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/locations`

auth: jwt (administrator | update_taxcore_credentials)

List all Locations for a business.

handlers: LocationsController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. Administrator bypass via requireBusinessAccess.

OUTPUTS

200

APIResponseDataList<LocationDTO>

All Locations for the business.

403

APIError

USER_FORBIDDEN — insufficient role or businessId mismatch.

POST `/api/v1/businesses/:businessId/locations`

auth: jwt (administrator | update_taxcore_credentials)

Create a new Location for the business. locationType defaults server-side; isDefault optionally promotes this Location to the business's default (existing default cleared atomically).

handlers: LocationsController.create

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`name` body string	yes	—	Display name (min 1 char).
`street` body string | null	no	—	Street address (min 1 char when set).
`city` body string | null	no	—	City (min 1 char when set).
`administrativeUnit` body string | null	no	—	State / province / region (min 1 char when set).
`country` body string | null	no	—	ISO 3166-1 alpha-2 code (e.g. 'VU').
`locationType` body 'Headquarters' | 'BranchOffice'	no	—	Defaults to BranchOffice in the repository when omitted.
`isDefault` body boolean	no	—	Promote this Location to the business default.

OUTPUTS

201

APIResponseDataObject<LocationDTO>

Newly created Location.

403

APIError

USER_FORBIDDEN — insufficient role or businessId mismatch.

422

APIError

VALIDATION_ERROR — schema violation.

GET `/api/v1/businesses/:businessId/locations/:locationId`

auth: jwt (administrator | update_taxcore_credentials)

Fetch a single Location scoped to the business.

handlers: LocationsController.getById

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Target Location UUID.

OUTPUTS

200

APIResponseDataObject<LocationDTO>

The Location.

404

APIError

LOCATION_NOT_FOUND.

PATCH `/api/v1/businesses/:businessId/locations/:locationId`

auth: jwt (administrator | update_taxcore_credentials)

Partial update of a Location. isDefault=true is routed through the dedicated setDefault path so the previous default is cleared atomically. status='inactive' soft-disables the Location.

handlers: LocationsController.update

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` path uuid	yes	—	Target Location UUID.
`name` body string	no	—	Display name (min 1 char).
`street` body string | null	no	—	Street address. Null clears.
`city` body string | null	no	—	City. Null clears.
`administrativeUnit` body string | null	no	—	Administrative unit. Null clears.
`country` body string | null	no	—	ISO 3166-1 alpha-2 code, or null.
`locationType` body 'Headquarters' | 'BranchOffice'	no	—	Updated location type.
`status` body 'active' | 'inactive'	no	—	Soft-enable/disable.
`isDefault` body boolean	no	—	Promote this Location to the business default (atomic swap).

OUTPUTS

200

APIResponseDataObject<LocationDTO>

Updated Location.

404

APIError

LOCATION_NOT_FOUND.

422

APIError

VALIDATION_ERROR — empty body or schema violation.

DELETE /api/v1/businesses/:businessId/locations/:locationId

auth: jwt (administrator | update_taxcore_credentials)

Hard-delete a Location (or soft-delete depending on repository impl — see ILocationRepository.remove).

handlers: LocationsController.remove

INPUTS

name	required	default	description
businessId path uuid	yes	—	Target business UUID.
locationId path uuid	yes	—	Target Location UUID.

OUTPUTS

204

void

Location removed.

404

APIError

LOCATION_NOT_FOUND.

vsmsconnect — API — mcp

← all endpoints · types ↗

GET `/api/v1/mcp/connection` auth: jwt

Return the public MCP server URL, transport, and tool list. Does NOT require an MCP token — only a valid JWT. Used by the app to display connection details before the user creates their first token.

handlers: McpController.getConnection

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<McpConnectionInfoDTO>

{ url, transport: 'streamable-http', tools: [...] } — URL is MCP_PUBLIC_URL/mcp.

POST /api/v1/mcp/tokens auth: jwt

Issue a 90-day RS256 personal-access token (typ='mcp') for AI agents. jti = McpTokens.McpTokenId UUID. Only SHA-256(rawToken) is persisted; the raw token is returned ONCE on creation and never again.

handlers: McpController.createToken

INPUTS

name	required	default	description
name body string	yes	—	Operator-facing label (1..100 chars), e.g. 'Claude Desktop on laptop'.

OUTPUTS

201

APIResponseDataObject<CreateMcpTokenResponse>

McpTokenDTO + raw `token` (RS256 JWT). Save the raw token — it will not be shown again.

422

APIError

VALIDATION_ERROR — schema violation.

GET /api/v1/mcp/tokens auth: jwt

List the caller's active MCP tokens (non-revoked, non-expired). Raw token hash is never exposed — only id, name, createdAt, lastUsedAt, expiresAt.

handlers: McpController.listTokens

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<McpTokenDTO>

Active tokens for the caller.

DELETE /api/v1/mcp/tokens/:tokenId auth: jwt

Soft-revoke an MCP token (sets RevokedAt). Idempotent — re-calling on an already-revoked token is a no-op.

handlers: McpController.revokeToken

INPUTS

name	required	default	description
`tokenId` path uuid	yes	—	McpTokens.McpTokenId to revoke (must belong to the caller).

OUTPUTS

204

`void`

Token revoked (or already-revoked).

404

APIError

MCP_TOKEN_NOT_FOUND — tokenId does not exist for this user.

GET /api/v1/mcp/agent-config auth: jwt

Return a Claude Desktop / mcp.json configuration snippet referencing the MCP server URL and a placeholder Authorization header. The raw token is not returned — the user must substitute the value they saved at creation time.

handlers: McpController.getAgentConfig

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<McpAgentConfigResult>

{ url, transport, activeToken: { id, name } | null, configSnippet: { mcpServers: { vsmsconnect: { url, headers: { Authorization: 'Bearer <your_mcp_token>' } } } } }.

404

APIError

MCP_TOKEN_NOT_FOUND — caller has no active MCP token; create one via POST /mcp/tokens first.

vsmsconnect — API — myob-tax-mappings

← all endpoints · types ↗

GET /api/v1/accounting/myob/tax-mappings

auth: jwt (administrator)

List all MYOB tax-mappings for the business (`Provider='myob'`). Mirrors `/accounting/sage/tax-mappings`.

handlers: MyobTaxMappingsController.list

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

MYOB mappings (active and inactive, active first).

POST /api/v1/accounting/myob/tax-mappings/auto-map

auth: jwt (administrator)

Pull MYOB tax codes + V-SDC `/status` and return match suggestions. Does NOT persist. Rate-limited (10/min open bucket).

handlers: MyobTaxMappingsController.autoMap

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<MyobTaxMappingAutoMapResponse>

Suggestions array.

401

APIError

MYOB_REAUTH_REQUIRED — MYOB refresh token rejected.

PUT `/api/v1/accounting/myob/tax-mappings` auth: jwt (administrator)

Create or update one MYOB tax-mapping. Fires `MYOB_TAX_MAPPING_UPDATED` audit.

handlers: MyobTaxMappingsController.upsert

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	MYOB tax-code (e.g. `GST`, `FRE`).
`providerTaxName` body string | null	no	—	Human-readable MYOB tax name.
`providerRate` body number | null	no	—	MYOB tax rate.
`vsdcTaxLabel` body string	yes	—	V-SDC label.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

MYOB_TAX_MAPPING_INVALID_LABEL — invalid `vsdcTaxLabel`.

DELETE `/api/v1/accounting/myob/tax-mappings/:providerTaxType`

auth: jwt (administrator)

Soft-delete a MYOB tax-mapping (`IsActive=0`). Idempotent.

handlers: MyobTaxMappingsController.remove

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	MYOB tax-code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH `/api/v1/accounting/myob/tax-mappings/:providerTaxType/restore`

auth: jwt (administrator)

Re-activate a deactivated MYOB tax-mapping (`IsActive=1`). Idempotent.

handlers: MyobTaxMappingsController.restore

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	MYOB tax-code to re-activate.

OUTPUTS

204

`empty`

No content.

vsmsconnect — API — myob

← all endpoints · types ↗

GET `/api/v1/accounting/myob/start` auth: jwt (administrator)

Begin a MYOB AccountRight Live OAuth2 authorisation flow. Generates PKCE + state, returns the MYOB authorisation URL. Rate-limited (10/min open bucket).

handlers: MyobAuthController.start

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<MyobStartResponse>

`{ authorizationUrl }`.

409

APIError

BUSINESS_CONTEXT_REQUIRED.

429

APIError

Rate limit exceeded.

GET `/api/v1/accounting/myob/callback` auth: none

MYOB OAuth callback. Validates state, exchanges the code, upserts a MyobConnections row per discovered company file, then 302-redirects (desktop) or serves a 200 text/html bridge page (mobile) opening `vsms://auth/myob/{success|failure}`.

handlers: MyobAuthController.callback

INPUTS

name	required	default	description
`code` query string	no	—	OAuth authorisation code returned by MYOB.
`state` query string	no	—	PKCE state issued by `/start`.
`error` query string	no	—	Set by MYOB on failure or user denial.

OUTPUTS

302

`redirect`

Desktop UA — redirect to `${MYOB_FRONTEND_WEB_BASE_URL}/auth/myob/{success|failure}`.

200

`text/html`

Mobile UA — HTML bridge to deep link with web fallback.

GET /api/v1/accounting/myob/status auth: jwt

MYOB connection summary — whether a connection exists, the file list, and the current `autoFiscaliseInvoices` flag.

handlers: MyobAuthController.status

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<MyobStatusResponse>

Connection summary including `connected`, `activeFile`, `files[]`, `autoFiscaliseInvoices`.

409

APIError

BUSINESS_CONTEXT_REQUIRED.

GET /api/v1/accounting/myob/files auth: jwt

List every MYOB company file authorised for the caller's business, each carrying `isActive` indicating the current sync target.

handlers: MyobAuthController.listFiles

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<MyobFileSummary>

Company file list.

POST /api/v1/accounting/myob/files/:myobFileId/activate

auth: jwt (administrator)

Switch the active MYOB company file for the caller's business. Fires `MYOB_FILE_ACTIVATED` audit.

handlers: MyobAuthController.activateFile

INPUTS

name	required	default	description
`myobFileId` path string	yes	—	MYOB company file ID (1–100 chars).

OUTPUTS

200

APIResponseDataObject<MyobFileSummary>

Newly active file.

404

APIError

File not found for this business.

DELETE /api/v1/accounting/myob/connection

auth: jwt (administrator)

Soft-deactivate all MYOB connections for the caller's business (`IsActive=0`). Rows are retained for history. Fires `MYOB_DISCONNECTED` audit.

handlers: MyobAuthController.disconnect

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<{ deactivatedCount: number }>

Number of rows deactivated.

vsmsconnect — API — print-agent

← all endpoints · types ↗

POST `/api/v1/print/agent/claim` auth: none

Print-agent first-run setup. Exchanges the one-time SetupToken (issued via `POST /print/tills/setup-token`) for a permanent TillToken — the SetupToken itself IS the credential for this single call. Returns the till + plaintext token (only time it is returned). Fires `TILL_CLAIMED` audit.

handlers: PrintAgentController.claim

INPUTS

name	required	default	description
`setupToken` body string	yes	—	One-time token from the setup-token endpoint.
`shopId` body string	yes	—	Business id surfaced to the agent.
`shopName` body string	yes	—	Business display name.
`tillName` body string	yes	—	Operator-supplied till name.
`machineId` body string	yes	—	Stable machine identifier (host fingerprint).
`posRegisterId` body string	no	—	AMICUS POS register id — recorded on `Tills.PosRegisterId` for register→till routing on push.

OUTPUTS

200

APIResponseDataObject<{ till: TillDTO; tillToken: string }>

Claimed till plus permanent TillToken (returned plaintext once).

422

APIError

VALIDATION_ERROR — setup token is invalid or has expired.

GET /api/v1/print/agent/jobs auth: tilltoken

Return PENDING print jobs for the authenticated till. Polled by the agent as a fallback to the SSE wake channel.

handlers: PrintAgentController.listJobs

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<PrintJobDTO>

PENDING jobs for the till.

401

APIError

Missing or invalid TillToken.

GET /api/v1/print/agent/jobs/stream auth: tilltoken SSE

Long-lived SSE channel — emits a `wake` event whenever a new PrintJob is published for this till so the agent can fetch + print without waiting for the next poll tick. Keep-alive `:keepalive` comment every 25 s to beat reverse-proxy idle timeouts.

handlers: PrintAgentController.streamJobs

INPUTS

No parameters.

OUTPUTS

200

text/event-stream

SSE stream — `wake` events on new PrintJob, keep-alive comments every 25 s.

401

APIError

Missing or invalid TillToken.

POST `/api/v1/print/agent/jobs/:id/ack` auth: tilltoken

Agent ack — mark a print job `PRINTED` (success) or `FAILED` (USB / printer error). The repository asserts the job belongs to the authenticated till and is still in a settable state.

handlers: PrintAgentController.ackJob

INPUTS

name	required	default	description
`id` path integer	yes	—	PrintJobId (coerced from string digits).
`status` body 'PRINTED' | 'FAILED'	no	`PRINTED`	Outcome (defaults to PRINTED).

OUTPUTS

204

`empty`

Acked.

401

APIError

Missing or invalid TillToken.

422

APIError

VALIDATION_ERROR — job not found or already processed.

POST `/api/v1/print/agent/heartbeat` auth: tilltoken

Periodic heartbeat — updates `Tills.IsOnline=1` and `LastSeenAt=now`. Sent by the agent on every poll tick / SSE reconnect.

handlers: PrintAgentController.heartbeat

INPUTS

No parameters.

OUTPUTS

> **204**
>
> `empty`
>
> Heartbeat recorded.

> **401**
>
> `APIError`
>
> Missing or invalid TillToken.

vsmsconnect — API — print

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/print/tills` auth: jwt (administrator)

List every registered till for the business — name, machine id, `IsDefault`, `IsOnline`, `LastSeenAt`, and optional `LocationId`.

handlers: PrintController.listTills

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID. `requireBusinessAccess` enforces this matches the JWT.

OUTPUTS

200

APIResponseDataList<TillDTO>

Till list.

PUT `/api/v1/businesses/:businessId/print/tills/:tillId/name` auth: jwt (administrator)

Rename a till. Fires `TILL_RENAMED` audit (resource type `TILL`).

handlers: PrintController.renameTill

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`tillId` path uuid	yes	—	Target till UUID.
`tillName` body string	yes	—	New display name (1–500 chars).

OUTPUTS

200

APIResponseDataObject<TillDTO>

Updated till.

404

APIError

Till not found for this business.

POST `/api/v1/businesses/:businessId/print/tills/:tillId/test` auth: jwt (administrator)

Dispatch a test PrintJob (`type='TEST'`) to the till. The agent picks it up on its next poll or via the SSE wake stream. Fires `TEST_PRINT_SENT` audit.

handlers: PrintController.testPrint

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`tillId` path uuid	yes	—	Target till UUID.

OUTPUTS

200

APIResponseDataObject<{ printJobId: number }>

Newly created print-job id.

404

APIError

Till not found for this business.

POST `/api/v1/businesses/:businessId/print/tills/setup-token` auth: jwt (administrator)

Generate a one-time setup token used by the print agent's first-run wizard to claim a permanent TillToken. Optionally binds the resulting till to a specific Location (TAXCORE-246) — single-location merchants omit and the claim resolves to the business default. Fires `TILL_SETUP_TOKEN_GENERATED` audit.

handlers: PrintController.generateSetupToken

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`locationId` body uuid	no	—	Optional Location to bind the till to. Validated against the business — invalid → 422 LOCATION_NOT_FOUND.

OUTPUTS

200

APIResponseDataObject<{ setupToken: string }>

Raw one-time setup token (only returned once).

422

APIError

LOCATION_NOT_FOUND — `locationId` does not belong to this business.

PATCH /api/v1/businesses/:businessId/print/default-till

auth: jwt (administrator)

Set the default till for the business. Future auto-print jobs that don't carry an explicit till routing key target this till. Fires `TILL_DEFAULT_SET` audit.

handlers: PrintController.setDefaultTill

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`tillId` body uuid	yes	—	Till UUID to mark default.

OUTPUTS

200

APIResponseDataObject<TillDTO>

Updated till.

404

APIError

Till not found.

GET /api/v1/businesses/:businessId/print/jobs

auth: jwt (administrator)

List print jobs for the business, optionally filtered by status.

handlers: PrintController.listPrintJobs

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`status` query 'PENDING' | 'PRINTED' | 'FAILED' | 'STALE'	no	—	Filter by `Status`.

OUTPUTS

200

APIResponseDataList<PrintJobDTO>

Print jobs.

POST `/api/v1/businesses/:businessId/print/jobs/:id/retry` auth: jwt (administrator)

Reset a `FAILED` or `STALE` print job back to `PENDING` so the agent will pick it up again. Fires `PRINT_JOB_RETRIED` audit.

handlers: PrintController.retryPrintJob

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.
`id` path integer	yes	—	PrintJobId (BIGINT IDENTITY, coerced from string digits).

OUTPUTS

200

APIResponseDataObject<PrintJobDTO>

Updated print job.

404

APIError

Print job not found or not retriable.

vsmsconnect — API — sage-tax-mappings

← all endpoints · types ↗

GET /api/v1/accounting/sage/tax-mappings

auth: jwt (administrator)

List all Sage tax-type to V-SDC tax-label mappings for the business (`Provider='sage'`). Mirrors `/accounting/xero/tax-mappings`.

handlers: SageAccountingTaxMappingsController.list

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

Sage mappings (active and inactive, active first).

POST /api/v1/accounting/sage/tax-mappings/auto-map

auth: jwt (administrator)

Pull Sage tax-rates + V-SDC `/status` and return match suggestions. Does NOT persist. Rate-limited (10/min open bucket).

handlers: SageAccountingTaxMappingsController.autoMap

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<SageTaxMappingAutoMapResponse>

Suggestions array.

401

APIError

REAUTH_REQUIRED — Sage refresh token rejected.

PUT `/api/v1/accounting/sage/tax-mappings` auth: jwt (administrator)

Create or update one Sage tax-mapping. Re-activates an inactive row on match. Fires `SAGE_TAX_MAPPING_UPDATED` audit (resource type `SAGE_CONNECTION`).

handlers: SageAccountingTaxMappingsController.upsert

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	Sage tax-type code.
`providerTaxName` body string | null	no	—	Human-readable Sage tax name.
`providerRate` body number | null	no	—	Sage tax rate.
`vsdcTaxLabel` body string	yes	—	V-SDC label — one of `F/N/P/E/T/A/B/C/D`.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

SAGE_TAX_MAPPING_INVALID_LABEL — invalid `vsdcTaxLabel`.

DELETE `/api/v1/accounting/sage/tax-mappings/:providerTaxType`

auth: jwt (administrator)

Soft-delete a Sage tax-mapping (`IsActive=0`). Idempotent.

handlers: SageAccountingTaxMappingsController.remove

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage tax-type code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH `/api/v1/accounting/sage/tax-mappings/:providerTaxType/restore`

auth: jwt (administrator)

Re-activate a deactivated Sage tax-mapping (`IsActive=1`). Idempotent.

handlers: SageAccountingTaxMappingsController.restore

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Sage tax-type code to re-activate.

OUTPUTS

204

`empty`

No content.

vsmsconnect — API — sage

← all endpoints · types ↗

GET `/api/v1/accounting/sage/start` auth: jwt (administrator)

Begin a Sage OAuth2 authorisation flow for the caller's business. Generates PKCE + state, persists in OAuthStates, returns the Sage authorisation URL. Rate-limited (10/min open bucket).

handlers: SageAccountingAuthController.start

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<SageStartResponse>

`{ authorizationUrl }`.

409

APIError

BUSINESS_CONTEXT_REQUIRED.

429

APIError

Rate limit exceeded.

GET `/api/v1/accounting/sage/callback` auth: none

Sage OAuth callback. Validates state, exchanges the code, upserts a single SageConnections row (Sage has one business per token — no tenant selection), then 302 redirects (desktop) or serves a 200 text/html bridge page (mobile) opening `vsms://auth/sage/{success|failure}`.

handlers: SageAccountingAuthController.callback

INPUTS

name	required	default	description
`code` query string	no	—	OAuth authorisation code returned by Sage.
`state` query string	no	—	PKCE state issued by `/start`.
`error` query string	no	—	Set by Sage on failure or user denial.

OUTPUTS

302

`redirect`

Desktop UA — redirect to `${SAGE_FRONTEND_WEB_BASE_URL}/auth/sage/{success|failure}`.

200

`text/html`

Mobile UA — HTML bridge page opening the native deep link with web fallback.

GET `/api/v1/accounting/sage/status` auth: jwt

Sage connection summary — whether a Sage connection exists, the connected Sage business, and the current `autoFiscaliseInvoices` flag.

handlers: SageAccountingAuthController.status

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<SageStatusResponse>

Connection summary.

409

APIError

BUSINESS_CONTEXT_REQUIRED.

DELETE `/api/v1/accounting/sage/connection` auth: jwt (administrator)

Soft-deactivate the Sage connection for the caller's business (`IsActive=0`). Rows are retained for history; the user may re-authorise via `/sage/start`. Fires `SAGE_DISCONNECTED` audit.

handlers: SageAccountingAuthController.disconnect

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<{ deactivatedCount: number }>

Number of rows deactivated.

vsmsconnect — API — sage200-tax-mappings

← all endpoints · types ↗

GET `/api/v1/accounting/sage200evolution/tax-mappings` auth: jwt (administrator)

List all Sage 200 Evolution tax-mappings for the business (`Provider='sage200evolution'`). The path-param `:id` everywhere else in this surface is the Sage `idTaxRate` (the provider tax type).

handlers: Sage200EvolutionTaxMappingsController.list

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

Sage 200 Evolution mappings.

POST `/api/v1/accounting/sage200evolution/tax-mappings/auto` auth: jwt (administrator)

One-off auto-map against the customer's on-premise Sage 200 Evolution database. Opens a direct MSSQL connection using the supplied credentials, runs `SELECT idTaxRate, Code, cFiscalTaxLabel FROM dbo.TaxRate`, and upserts a proposed mapping for every row not already mapped. Distinct from the `auto-map` pattern used by Xero/Sage/MYOB — this endpoint reads the customer's own DB and persists proposals immediately.

handlers: Sage200EvolutionTaxMappingsController.autoMap

INPUTS

name	required	default	description
`host` body string	yes	—	Sage 200 Evolution DB host (customer's on-prem MSSQL server).
`port` body integer	yes	—	MSSQL port (1–65535). Coerced from string if needed.
`database` body string	yes	—	Database name.
`user` body string	yes	—	SQL login.
`password` body string	yes	—	SQL password.

OUTPUTS

200

`APIResponseDataObject<{ proposed: TaxRateMappingDTO[]; skipped: object[]; alreadyMapped: number }>`

`proposed[]` — newly upserted proposals; `skipped[]` — rows whose `cFiscalTaxLabel` is missing or not a valid V-SDC label; `alreadyMapped` — count of rows that already have an active mapping.

502

`APIError`

External Sage DB connection failed.

PUT `/api/v1/accounting/sage200evolution/tax-mappings/:id` auth: jwt (administrator)

Upsert a single Sage 200 Evolution mapping. `:id` is the Sage `idTaxRate` (provider tax type). Body carries the V-SDC label + optional metadata.

handlers: Sage200EvolutionTaxMappingsController.upsert

INPUTS

name	required	default	description
`id` path string	yes	—	Sage `idTaxRate` — the provider tax type.
`providerTaxName` body string | null	no	—	Human-readable Sage tax name.
`providerRate` body number | null	no	—	Sage tax rate percent.
`vsdcTaxLabel` body string	yes	—	V-SDC label.
`vsdcRate` body number | null	no	—	V-SDC rate percent.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping.

422

APIError

Invalid V-SDC label.

DELETE `/api/v1/accounting/sage200evolution/tax-mappings/:id`

auth: jwt (administrator)

Soft-delete a Sage 200 Evolution mapping (`IsActive=0`). Idempotent.

handlers: Sage200EvolutionTaxMappingsController.remove

INPUTS

name	required	default	description
`id` path string	yes	—	Sage `idTaxRate`.

OUTPUTS

204

`empty`

No content.

POST `/api/v1/accounting/sage200evolution/tax-mappings/:id/restore`

auth: jwt (administrator)

Re-activate a deactivated Sage 200 Evolution mapping (`IsActive=1`). Note: uses POST (unlike the Xero/Sage/MYOB restore endpoints which use PATCH). Idempotent.

handlers: Sage200EvolutionTaxMappingsController.restore

INPUTS

name	required	default	description
`id` path string	yes	—	Sage `idTaxRate`.

OUTPUTS

204

`empty`

No content.

vsmsconnect — API — sync-from-date

← all endpoints · types ↗

PATCH `/api/v1/accounting/sync-from-date` auth: jwt (administrator)

Per-business sync lower-bound date — Administrator-only. The stored epoch-ms lower bound is read by all sync paths (manual Sync Now + scheduled workers for Xero/Sage/MYOB) and passed as modifiedSince to the accounting provider. Setting to null restores the default 'full history' behaviour. Webhooks are never filtered. Surfaced on GET /accounting/{xero|sage|myob}/status so the app can render the date picker pre-populated.

handlers: SyncFromDateController.update

INPUTS

Request body type: SyncFromDateUpdateBody

name	required	default	description
`syncFromDate` body ISO-8601 datetime | null	yes	—	Lower-bound date for incoming syncs. Null clears the lower bound.

OUTPUTS

200

`APIResponseDataObject<SetSyncFromDateResult>`

Persisted result (typically { syncFromDate: epochMs | null }).

401

`APIError`

AUTH_UNAUTHORIZED — caller has no JWT user.

409

`APIError`

BUSINESS_CONTEXT_REQUIRED — no businessId on the JWT.

422

`APIError`

VALIDATION_ERROR — non-ISO datetime.

vsmsconnect — API — unmapped-tax-types

← all endpoints · types ↗

GET `/api/v1/businesses/:businessId/accounting/unmapped-tax-types`

auth: jwt (administrator | view_invoices | submit_invoices)

List provider tax codes that have arrived on invoices without a confirmed mapping. Populated fire-and-forget by sync services and webhook workers on every invoice ingestion pass. Each row includes providerTaxName and providerRate (nullable). Rows that now have an active TaxRateMappings entry are excluded server-side via NOT EXISTS join — confirming a mapping silently removes the entry; deactivating a mapping makes it reappear. Used by the home-screen banner and the Integrations nav badge to notify admins.

handlers: UnmappedTaxTypesController.list

INPUTS

name	required	default	description
`businessId` path uuid	yes	—	Target business UUID.

OUTPUTS

200

APIResponseDataList<UnmappedTaxTypeDTO>

Provider tax codes awaiting admin mapping. Each row: { id, provider, providerTaxType, providerTaxName, providerRate, firstSeenAt, lastSeenAt }.

vsmsconnect — API — users

← all endpoints · types ↗

GET `/api/v1/users` auth: jwt (administrator | manage_users)

List all users belonging to the authenticated caller's business (businessId taken from the JWT, not the URL).

handlers: UsersController.list

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<UserAdminDTO>

All users for the JWT's businessId. Includes totalRows.

POST `/api/v1/users` auth: jwt (administrator | manage_users)

Create a new user inside the caller's business. Argon2-hashes the password, derives username from the email local part, inserts into Users + UserRoles, and fires USER_CREATED + welcome email (fire-and-forget).

handlers: UsersController.create

INPUTS

name	required	default	description
`firstName` body string	yes	—	Given name (min 1 char).
`lastName` body string	yes	—	Family name (min 1 char).
`email` body string (email)	yes	—	Email address — must be globally unique.
`roles` body UserRole[]	yes	—	At least one role from the UserRole enum.
`password` body string	yes	—	Plaintext password (min 8 chars).

OUTPUTS

201

APIResponseDataObject<UserAdminDTO>

Newly created user.

409

APIError

AUTH_EMAIL_TAKEN — email already registered.

422

APIError

VALIDATION_ERROR — schema violation.

PATCH `/api/v1/users/:userId` auth: jwt (administrator | manage_users)

Partial update of a user inside the caller's business. Patch may include roles, isActive, firstName, lastName, email, and/or password. Admin-removal safeguards prevent removing your own Administrator role or removing the last administrator. Fires USER_DEACTIVATED on isActive=false, USER_UPDATED otherwise.

handlers: UsersController.update

INPUTS

name	required	default	description
`userId` path uuid	yes	—	Target user UUID — must belong to the caller's business.
`roles` body UserRole[]	no	—	Replacement role set (min 1).
`isActive` body boolean	no	—	Activate/deactivate the user.
`firstName` body string	no	—	Given name (min 1 char).
`lastName` body string	no	—	Family name (min 1 char).
`email` body string (email)	no	—	Replacement email address.
`password` body string	no	—	Replacement password (min 8 chars, argon2-hashed server-side).

OUTPUTS

200

APIResponseDataObject<UserAdminDTO>

Canonical post-update user record (re-fetched from the business set).

404

APIError

USER_NOT_FOUND — userId not in caller's business.

422

APIError

USER_CANNOT_REMOVE_OWN_ADMIN | USER_CANNOT_REMOVE_LAST_ADMIN | VALIDATION_ERROR (empty body or schema violation).

vsmsconnect — API — well-known

← all endpoints · types ↗

GET /.well-known/apple-app-site-association

auth: none

iOS Universal Links association file. Returns a JSON payload listing the app's TeamID + bundleIdentifier under `applinks.details` with path scope `/go*`. Served unauthenticated and OUTSIDE the /api/v1 prefix — the OS fetches it directly over HTTPS. TeamID is a deploy-time placeholder.

handlers: WellKnownRoute inline handler

INPUTS

No parameters.

OUTPUTS

200 application/json

```
{ applinks: { apps: string[]; details: { appID: string; paths: string[] }[] } }
```

Apple-formatted association payload (application/json). appID format: <TEAM_ID>.<bundleIdentifier>.

GET /.well-known/assetlinks.json

auth: none

Android App Links assetlinks. Returns a JSON array delegating common.handle_all_urls to the app's android_app namespace with package_name + sha256_cert_fingerprints. Served unauthenticated and OUTSIDE the /api/v1 prefix. Package name and fingerprint are deploy-time placeholders.

handlers: WellKnownRoute inline handler

INPUTS

No parameters.

OUTPUTS

200 application/json

```
Array<{ relation: string[]; target: { namespace: string; package_name: string; sha256_cert_fingerprints: string[] } }>
```

Android-formatted assetlinks payload (application/json).

vsmsconnect — API — xero-tax-mappings

← all endpoints · types ↗

GET /api/v1/accounting/xero/tax-mappings

auth: jwt (administrator)

List all Xero tax-type to V-SDC tax-label mappings for the business (active and inactive). Active rows ordered first, then alphabetically. `isActive: boolean` per row so the admin UI can surface a Restore action on soft-deleted entries.

handlers: XeroTaxMappingsController.list

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<TaxRateMappingDTO>

Mappings for `Provider='xero'`.

POST /api/v1/accounting/xero/tax-mappings/auto-map

auth: jwt (administrator)

Fetch Xero `/TaxRates` from the active tenant + V-SDC `/status`, then return match suggestions with `needsReview` flags + alternatives. Does NOT persist — the admin confirms each suggestion via PUT. Rate-limited (10/min open bucket).

handlers: XeroTaxMappingsController.autoMap

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<XeroTaxMappingAutoMapResponse>

Suggestions array with `providerTaxType`, `suggestedVsdcLabel`, `alternatives[]`, `needsReview`.

401

APIError

REAUTH_REQUIRED — Xero refresh token rejected.

429

APIError

Rate limit exceeded (caller or Xero).

PUT `/api/v1/accounting/xero/tax-mappings` auth: jwt (administrator)

Create or update one Xero tax-type to V-SDC label mapping (MERGE on (businessId, 'xero', providerTaxType)). Re-activates an existing inactive row instead of inserting a duplicate. Fires `XERO_TAX_MAPPING_UPDATED` audit.

handlers: XeroTaxMappingsController.upsert

INPUTS

name	required	default	description
`providerTaxType` body string	yes	—	Xero tax-type code (e.g. `OUTPUT2`, `ZERORATED`). Trimmed, 1–100 chars.
`providerTaxName` body string | null	no	—	Human-readable Xero tax name for display.
`providerRate` body number | null	no	—	Xero rate percent.
`vsdcTaxLabel` body string	yes	—	Target V-SDC tax label (must be one of `F/N/P/E/T/A/B/C/D`).
`vsdcRate` body number | null	no	—	V-SDC rate percent for display.

OUTPUTS

200

APIResponseDataObject<TaxRateMappingDTO>

Upserted mapping row.

422

APIError

XERO_TAX_MAPPING_INVALID_LABEL — `vsdcTaxLabel` is not a valid V-SDC label.

DELETE `/api/v1/accounting/xero/tax-mappings/:providerTaxType`

auth: jwt (administrator)

Soft-delete a Xero tax mapping — sets `IsActive=0`. Row is retained for history and can be re-activated via the restore endpoint. Idempotent.

handlers: XeroTaxMappingsController.remove

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Xero tax-type code to deactivate.

OUTPUTS

204

`empty`

No content.

PATCH `/api/v1/accounting/xero/tax-mappings/:providerTaxType/restore`

auth: jwt (administrator)

Re-activate a previously soft-deleted Xero tax mapping (`IsActive=0` → `1`). Idempotent.

handlers: XeroTaxMappingsController.restore

INPUTS

name	required	default	description
`providerTaxType` path string	yes	—	Xero tax-type code to re-activate.

OUTPUTS

204

`empty`

No content.

vsmsconnect — API — xero-webhook

← all endpoints · types ↗

POST `/webhooks/xero` auth: hmac (x-xero-signature)

Xero-initiated webhook delivery. Verifies HMAC-SHA256 over the raw request body using `XERO_WEBHOOK_SIGNING_KEY` (constant-time compared to the base64 `x-xero-signature` header). Valid signature → fan each event to the `xero:webhook:events` Redis Stream and respond `200` within Xero's 5-second ack budget. Xero's intent-to-receive handshake (`events: []`) flows through the same code path. NOT mounted under `/api/v1`.

handlers: XeroWebhookController.handle

INPUTS

name	required	default	description
`x-xero-signature` header string	yes	—	Base64-encoded HMAC-SHA256 of the raw request body.
`events` body XeroWebhookEvent[]	no	—	Array of event objects (`tenantId`, `tenantType`, `eventCategory`, `eventType`, `resourceId`, `resourceUrl`, `eventDateUtc`, `eventSequence`). Empty array is the intent-to-receive ping.
`firstEventSequence` body integer	no	—	Sequence number of the first event in the batch.
`lastEventSequence` body integer	no	—	Sequence number of the last event in the batch.
`entropy` body string	no	—	Random salt included by Xero.

OUTPUTS

200

`empty`

Valid signature — events XADDed to Redis Stream, empty body.

401

APIError

XERO_WEBHOOK_SIGNATURE_INVALID — HMAC mismatch.

503

APIError

XERO_WEBHOOK_NOT_CONFIGURED — `XERO_WEBHOOK_SIGNING_KEY` is unset (dev safe default).

vsmsconnect — API — xero

← all endpoints · types ↗

GET `/api/v1/accounting/xero/start` auth: jwt (administrator)

Begin a Xero OAuth2 authorisation flow for the caller's business. Generates a PKCE verifier + state, persists them in OAuthStates, and returns the Xero authorisation URL the app/browser must redirect to. Rate-limited (10/min open bucket).

handlers: XeroAuthController.start

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<XeroStartResponse>

`{ authorizationUrl }` — the Xero hosted-consent URL.

409

APIError

BUSINESS_CONTEXT_REQUIRED — caller has not completed business registration.

429

APIError

Rate limit exceeded.

GET `/api/v1/accounting/xero/callback` auth: none

OAuth callback target hit by the user's browser after Xero consent. Validates the state (constant-time), exchanges the authorisation code, upserts XeroConnections rows for every tenant returned, and either 302-redirects (desktop) or serves a 200 text/html bridge page (mobile UA) that opens the native `vsms://auth/xero/...` deep link.

handlers: XeroAuthController.callback

INPUTS

name	required	default	description
`code` query string	no	—	OAuth authorisation code returned by Xero on success.
`state` query string	no	—	Opaque PKCE state value previously issued by `/start`. Constant-time compared against OAuthStates row.
`error` query string	no	—	Set by Xero when the user denies consent or the flow fails. Triggers the failure bridge / failure deep link.
`error_description` query string	no	—	Human-readable error message from Xero (logged, sanitised before being exposed).

OUTPUTS

302

`redirect`

Desktop User-Agent — Location header points at `${XERO_FRONTEND_WEB_BASE_URL}/auth/xero/{success|failure}` carrying tenant name or sanitised error code.

200

`text/html`

Mobile User-Agent — HTML bridge page that tries `XERO_FRONTEND_SUCCESS_URL` / `XERO_FRONTEND_FAILURE_URL` deep link then falls through to the web URL after 900 ms.

GET /api/v1/accounting/xero/status auth: jwt

Connection summary for the caller's business — whether a Xero connection exists, the active tenant, the list of authorised tenants (no tokens), and the current `autoFiscaliseInvoices` flag.

handlers: XeroAuthController.status

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<XeroStatusResponse>

Connection summary including `connected`, `activeTenant`, `tenants[]`, and `autoFiscaliseInvoices`.

409

APIError

BUSINESS_CONTEXT_REQUIRED — caller has no businessId on the JWT.

GET /api/v1/accounting/xero/tenants auth: jwt

List every Xero tenant authorised for the caller's business. Each entry carries `isActive` to indicate which tenant is the current target of sync / webhooks.

handlers: XeroAuthController.tenants

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataList<XeroTenantSummary>

Tenant list with `tenantName`, `tenantId`, `isActive`.

POST `/api/v1/accounting/xero/tenants/:xeroTenantId/activate` auth: jwt (administrator)

Switch the active Xero tenant for the caller's business. Flips `IsActive` on XeroConnections rows so subsequent sync / webhook traffic uses the chosen tenant.

handlers: XeroAuthController.activateTenant

INPUTS

name	required	default	description
`xeroTenantId` path uuid	yes	—	Xero tenant GUID to mark active.

OUTPUTS

200

APIResponseDataObject<XeroTenantSummary>

Newly active tenant summary.

404

APIError

Tenant not found for this business.

DELETE `/api/v1/accounting/xero/connection` auth: jwt (administrator)

Hard-disconnect Xero for the caller's business — deletes every XeroConnections row, drops encrypted tokens, fires `XERO_DISCONNECTED` audit.

handlers: XeroAuthController.disconnect

INPUTS

No parameters.

OUTPUTS

200

APIResponseDataObject<{ deletedCount: number }>

Number of connection rows deleted.

vsmsconnect — API — Types

← all endpoints · 109 types

Index

Definitions

AccountingInvoice TYPE

packages\dtos\src\dtos\accounting\accountingInvoice.dto.ts

references: AccountingInvoiceLineItem, AccountingInvoicePayment, AccountingInvoiceType

```
export type AccountingInvoice = {
  /** Provider's unique identifier for the invoice. */
  id: string;
  invoiceNumber: string | null;
  reference: string | null;
  type: AccountingInvoiceType;
  /** Provider-native status string (e.g. Xero "DRAFT", "AUTHORISED", "PAID"). */
  status: string;
  contactId: string | null;
  contactName: string | null;
  /**
   * Tax / registration number from the contact record, when the provider
   * exposes it. Maps to V-SDC's `buyerId` — do NOT silently null this
   * downstream.
   */
  contactTaxNumber: string | null;
  /**
   * Pre-formatted single-line address string built from the contact's
   * address record. Null when the provider didn't supply one.
   */
  contactAddress: string | null;
  /** ISO-8601 date string (YYYY-MM-DD). */
  issueDate: string | null;
  /** ISO-8601 date string (YYYY-MM-DD). */
  dueDate: string | null;
  /** ISO-4217 currency code. */
  currencyCode: string;
  subTotal: number;
  totalTax: number;
  total: number;
  amountDue: number;
  amountPaid: number;
  lineItems: AccountingInvoiceLineItem[];
  /**
   * Payments recorded against the invoice in the provider's books.
   *
   * Wired for Xero — populated from `Invoice.Payments`. Sage and MYOB
   * adapters currently emit `[]` until their payment-extraction is wired.
   * The downstream normaliser uses this array to decompose partially-paid
   * invoices into per-payment ADVANCE fiscal records (TAXCORE-139).
   */
  payments: AccountingInvoicePayment[];
  /** ISO-8601 timestamp of the most recent upstream change. */
  updatedAt: string | null;
  /**
```

```
     * For credit notes (`ACCRECCREDIT`), the provider's invoice-number (or UUID)
     * of the original invoice this credit note cancels. Maps to V-SDC's
     * `referentDocumentNumber`. Null on normal sales.
     */
    referentInvoiceId: string | null;
    /**
     * ISO-8601 date (YYYY-MM-DD) of the referenced original invoice. Maps
     * to V-SDC's `referentDocumentDT`. Null on normal sales.
     */
    referentIssueDate: string | null;
    /**
     * Provider-native status of the referenced original invoice when this row
     * is a credit note. Used to distinguish a standard refund (referent PAID,
     * `buyerId` = buyer TIN) from a cancellation (referent AUTHORISED,
     * `buyerId` = seller TIN). Null on normal sales or when the provider
     * does not embed the linked invoice's status.
     */
    referentInvoiceStatus: string | null;
    /**
     * Provider-specific invoice type override. When set (e.g. 'ADVANCE' for
     * Xero Prepayments, 'PROFORMA' for Xero Quotes), the normaliser passes
     * the value through to V-SDC. When null, the normaliser defaults to
     * `NORMAL`.
     */
    invoiceType?: string | null;
    /**
     * On-premise agent's intent for auto-fiscalisation. When `true`, the
     * cloud's auto-fiscalise gate bypasses its default PROFORMA exclusion
     * (still subject to the business-level `Businesses.AutoFiscaliseInvoices`
     * switch). When `false` or unset, PROFORMA invoices land unfiscalised so
     * the admin can review and dispatch from the Quotes tab.
     *
     * AMICUS source: `true` for quotations in ``'Quotation'`` (locked) status,
     * `false` for ``'Entry'`` (draft) status. Non-AMICUS providers leave it
     * unset — current Xero/Sage/MYOB exclusion of PROFORMA remains the
     * default behaviour.
     */
    autoFiscalise?: boolean;
    /**
     * Local primary key of the source POS row. AMICUS-only today
     * (`sale.number` for sales, `ReverseSale.Id` for refunds). Persisted
     * into `Invoices.PosSaleId` by the TAXCORE-207 push-endpoint extension.
     * Unused by every non-AMICUS path.
     */
    posSaleId?: string;
    /**
     * Register / lane the sale was rung up on. AMICUS-only today
     * (`sale.till_number` / `ReverseSale.till_number`). The TAXCORE-207
     * extension uses this to route receipt printing to the matching till.
     */
    posRegisterId?: string;
```

```
  /**
   * Refunds only — the original sale's `posSaleId`. Used by the TAXCORE-207
   * extension to resolve the V-SDC referent via
   * `(BusinessId, Source='amicus', PosSaleId=originalPosSaleId)`.
   */
  originalPosSaleId?: string;
  /**
   * Per-payment refund linkage (TAXCORE-293, AMICUS reverse-payment flow).
   * When set on an `ACCRECCREDIT` invoice, identifies the SPECIFIC prior
   * payment on the original sale that this refund reverses, by its
   * deposit-shape `externalId` (e.g. ``'683577001:deposit:0'``). The backend
   * resolves the corresponding `InvoicePayments` row via
   * `(Invoices.PosSaleId = originalPosSaleId, InvoicePayments.ExternalPaymentId = this)`
   * and routes the refund through the per-payment `createRefundInvoice`
   * primitive so the V-SDC submission inherits the source payment's
   * `FiscalInvoiceNumber` as the chain referent.
   *
   * Absent / null on whole-sale refunds (the existing AMICUS `ReverseSale`
   * path) — those route through the regular refund pipeline with no
   * per-payment anchor.
   */
  originalPaymentExternalId?: string | null;
  /**
   * Cashier / operator identifier of the person who rang the sale up.
   * Persisted into `Invoices.CashierId` and sent verbatim as the V-SDC
   * `cashier` field on the fiscal-invoice request. Free-form string —
   * AMICUS sends the numeric `dbo.sale.sales_person` value cast to string;
   * other providers can map to a TIN / staff number / email as appropriate.
   * When null, the consumer falls back to ``'Admin''.
   */
  cashierId?: string | null;
  /**
   * Buyer email address snapshotted from the source accounting / POS system
   * (TAXCORE-308). Sage 200 Evolution resolves it via a LEFT JOIN to
   * `dbo.Client` on `InvNum.AccountID = Client.DCLink`. Persisted into
   * `Invoices.BuyerEmail` so a fiscalised invoice carries `buyerEmail`.
   * Optional — undefined / null on providers whose normaliser does not yet
   * surface it (Xero / AMICUS / MYOB are explicit follow-ups). Existing
   * constructors are unaffected.
   */
  buyerEmail?: string | null;
};
```

AccountingInvoiceLineItem TYPE

packages\dtos\src\dtos\accounting\accountingInvoice.dto.ts

```
export type AccountingInvoiceLineItem = {
  description: string;
  quantity: number;
  unitAmount: number;
  taxAmount: number;
  lineAmount: number;
  accountCode: string | null;
  /** Provider-specific tax code (matches TaxRate.taxType). */
  taxType: string | null;
  /** Human-readable tax name resolved by the agent at push time (e.g. "VAT 15%"). */
  taxName?: string | null;
};
```

AccountingInvoicePayment TYPE

packages\dtos\src\dtos\accounting\accountingInvoice.dto.ts

references: AccountingInvoiceTender

```
export type AccountingInvoicePayment = {
  /** Provider's stable identifier for the payment (e.g. Xero PaymentID). */
  externalId: string;
  /**
   * Either a date-only ISO-8601 string (`YYYY-MM-DD`) — the historical Xero /
   * Sage / MYOB shape — or a full ISO-8601 datetime with an explicit offset
   * (e.g. `"2026-06-10T13:00:00+11:00"`). The full-datetime form is used by the
   * AMICUS sales agent so V-SDC's printed `POS Time` reflects the operator's
   * wall clock instead of UTC midnight; the AMICUS backend normaliser parses
   * either form via `dateOnlyToEpochMs`. Null when the provider didn't supply
   * one.
   */
  date: string | null;
  /** Payment amount in invoice currency. */
  amount: number;
  /** Free-form reference / description from the provider. */
  reference: string | null;
  /**
   * Xero `Account.BankAccountType` enriched per-payment via a follow-up
   * `GET /Payments/{PaymentID}` call. The values come back as
   * `'CREDITCARD' | 'BANK' | 'PAYPAL' | 'NONE'`. Null when the provider
   * doesn't surface an Account, when enrichment failed (Account not found,
   * 4xx other than 404), or for non-Xero providers (Sage / MYOB synthetic).
   * Consumed by `resolveXeroPaymentType` in the normaliser to derive the
   * V-SDC `PaymentType` for each fiscal event (TAXCORE-174 Phase 1).
   */
  accountBankType: string | null;
  /**
   * Xero `Account.Code` — tenant-defined account identifier (e.g. `090`).
   * Diagnostic / future admin-override mapping key. Null when absent.
   */
  accountCode: string | null;
  /**
   * Xero `Account.Name` — used as a fallback signal when `accountBankType`
   * is `NONE` (e.g. "Cash on Hand" → `CASH`). Diagnostic for everything
   * else.
   */
  accountName: string | null;
  /**
   * Split-tender breakdown (AMICUS multi-tender — TAXCORE-243). One entry per
   * `dbo.payment` row. When non-empty the normaliser resolves each
   * `paymentTypeCode` and the consumer emits one V-SDC `payment[]` entry per
   * tender on a single fiscal record. Undefined/empty → the scalar
   * `accountCode` / `paymentType` path is used (Xero/Sage/MYOB + AMICUS
   * single-tender).
```

```
   */
  tenders?: AccountingInvoiceTender[];
};
```

AccountingInvoiceTender TYPE

packages\dtos\src\dtos\accounting\accountingInvoice.dto.ts

```
/**
 * Shared representation of an invoice pulled from an accounting system.
 * Distinct from the fiscalisation-facing `InvoiceDTO` — this is the
 * upstream, pre-fiscalisation shape as the accounting system sees it.
 */
/**
 * Normalised payment recorded against an invoice in the provider's books.
 *
 * Each entry corresponds to one upstream payment row; the downstream
 * normaliser uses them to decompose partially-paid invoices into
 * one ADVANCE fiscal record per payment (TAXCORE-139).
 *
 * Adapters that have nothing to surface (Sage, MYOB pending integration)
 * emit an empty array and the invoice flows through unchanged.
 */
/**
 * One tender within a split-tender payment (AMICUS multi-tender — TAXCORE-243).
 *
 * AMICUS records ONE `dbo.sale` row with N rows in `dbo.payment` (one per
 * tender). When present on an `AccountingInvoicePayment`, the normaliser
 * resolves each `paymentTypeCode` against the SAME per-business
 * `paymentTypeByCode` map used for the scalar `accountCode`, producing one
 * V-SDC wire `payment[]` entry per tender on a single fiscal record.
 *
 * Optional end-to-end: when undefined/empty the scalar `accountCode` /
 * `paymentType` path is used (backward compatible for Xero/Sage/MYOB and
 * AMICUS single-tender sales).
 */
export type AccountingInvoiceTender = {
  /** Tender amount in invoice currency. */
  amount: number;
  /** Provider tender code (e.g. AMICUS `dbo.payment.payment_type` as string). */
  paymentTypeCode: string;
  /** Free-form reference (e.g. cheque number). Null when none. */
  reference?: string | null;
};
```

AccountingInvoiceType TYPE

packages\dtos\src\dtos\accounting\accountingInvoice.dto.ts

```
/**
 * Xero distinguishes four document kinds:
 *  - `ACCREC` — sales invoice (fiscalisable).
 *  - `ACCRECCREDIT` — credit note against a sale (fiscalisable as REFUND).
 *  - `ACCPAY` — supplier bill (not fiscalisable; we never ingest these).
 *  - `ACCPAYCREDIT` — credit note against a bill (not fiscalisable).
 *
 * Narrower providers (Sage, MYOB) can reuse whichever codes they support;
 * the normalisation step decides whether a row is eligible for V-SDC.
 */
export type AccountingInvoiceType =
  | 'ACCREC'
  | 'ACCPAY'
  | 'ACCRECCREDIT'
  | 'ACCPAYCREDIT';
```

AgentHeartbeat TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

```
/**
 * Operator-visibility heartbeat reported by the sales agent (TAXCORE-321).
 * POSTed periodically to `POST /accounting/agent/heartbeat`, stored per API key
 * in Redis with a TTL, and surfaced (freshest per provider) on the push-status
 * response. All fields are loose — the agent's status object is the source of
 * truth and may grow over time, so the backend validates leniently and the app
 * reads defensively.
 */
export type AgentHeartbeat = {
  /** Epoch ms of the agent's most recent push attempt, or null. */
  lastPushAt: number | null;
  /** Outcome of the agent's most recent cycle. */
  lastPushOutcome: 'success' | 'transient' | 'persistent_outage' | null;
  /** Consecutive transient failures the agent has seen (0 = healthy). */
  consecutiveFailures: number;
  /** True while the agent is in a persistent-outage state (drives the red badge). */
  persistentOutageActive: boolean;
  /** Approximate unpushed-row counts per stream (`sales` / `refunds` / …). */
  backlogEstimate: Record<string, number>;
  /** Epoch ms the backend received this heartbeat (stamped server-side). */
  receivedAt: number;
};
```

APIError CLASS

packages\dtos\src\entities\APIError.ts

references: APIErrorProps, APIResponse

```
export class APIError<T> extends Error {
  public status: number;
  public response: APIResponse<T>;
  public data: any;
  /**
   * If true, the next middleware will still be called
   */
  public soft = false;

  protected parent: Error;

  /**
   *
   * @param param.soft If true, the next middleware will still be called
   */
  constructor({ status, message, response, data, err, soft }: APIErrorProps<T>) {
    super();

    if (status) this.status = status;
    if (soft) this.soft = soft;
    if (response) this.response = response;
    if (data) this.data = data;

    if (err) {
      this.parent = err;
      this.message = err.message;
      this.stack = err.stack;
    }

    if (message) {
      this.message = message;
    }
  }

  public toDTO(): any {
    const dto = Object.keys(this)
      .filter((key) => !['stack', 'soft'].includes(key))
      .reduce(
        (acc, key) => ({ ...acc, [key]: this[key] }),
        {} as APIErrorProps<T>,
      );
    return dto;
  }
}
```

APIErrorProps TYPE

packages\dtos\src\entities\APIError.ts

references: APIResponse

```
type APIErrorProps<T> = {
  status?: number;
  message?: string;
  response?: APIResponse<T>;
  data?: any;
  err?: Error;
  soft?: boolean;
};
```

ApiKeyDTO TYPE

packages\dtos\src\dtos\apiKey.dto.ts

references: ConnectorType

```
export type ApiKeyDTO = {
  id: string;
  label: string;
  keyPrefix: string;
  scope: ConnectorType | null;
  /** The Location this key is scoped to (TAXCORE-246). NULL = business default. */
  locationId: string | null;
  createdAt: number;
  revokedAt: number | null;
};
```

APIResponse CLASS

packages\dtos\src\entities\APIResponse.ts

references: APIResponseConstructor, APIResponseDTO

```
export class APIResponse<T> {
  public error: boolean;
  public status: number;
  public message?: string;
  public data?: T | T[];
  public hasMore?: boolean;
  public totalRows?: number;
  public availableSources?: string[];

  constructor(obj?: APIResponseConstructor<T>) {
    this.error = obj?.error ?? false;
    this.status =
      obj?.status ??
      (this.error ? httpStatus.INTERNAL_SERVER_ERROR : httpStatus.OK);
    this.message = obj?.message;
    this.data = obj?.data;
    this.hasMore = obj?.hasMore;
    this.totalRows = obj?.totalRows;
    this.availableSources = obj?.availableSources;
  }

  public toDTO(): APIResponseDTO<T> {
    return {
      status: this.status,
      error: this.error,
      message: this.message,
      totalRows: this.totalRows,
      availableSources: this.availableSources,
      data: Array.isArray(this.data)
        ? {
            object: 'list',
            data: this.data,
            n: this.data.length,
            hasMore: this.hasMore ?? false,
            url: '',
          }
        : {
            object: 'object',
            data: this.data,
            hasMore: this.hasMore ?? false,
            url: '',
          },
    };
  }
```

```
  get isError(): boolean {
    return this.error;
  }
}
```

APIResponseConstructor TYPE

packages\dtos\src\entities\APIResponse.ts

references: APIResponseDTO

```
export type APIResponseConstructor<T> = Partial<
  Omit<APIResponseDTO<T>, 'data'>
> & {
  data?: T | T[];
  hasMore?: boolean;
  totalRows?: number;
  availableSources?: string[];
};
```

APIResponseDataList TYPE

packages\dtos\src\types\APIResponseDTO.ts

```
export type APIResponseDataList<T> = {
  object: 'list';
  data: T[];
  n: number;
  hasMore: boolean;
  url: string;
};
```

APIResponseDataObject TYPE

packages\dtos\src\types\APIResponseDTO.ts

```
export type APIResponseDataObject<T> = {
  object: 'object';
  data: T | undefined;
  hasMore: boolean;
  url: string;
};
```

APIResponseDTO TYPE

packages\dtos\src\types\APIResponseDTO.ts

references: APIResponseDataList, APIResponseDataObject

```
export type APIResponseDTO<T = unknown> = {
  status: number;
  error: boolean;
  message?: string;
  data?: APIResponseDataList<T> | APIResponseDataObject<T>;
  totalRows?: number;
  /**
   * Distinct provider/source values across the resource set, independent of
   * the current page or applied filters. Currently populated only by the
   * paginated invoices list endpoint so the home Provider dropdown can list
   * every source that has ever produced a row — including disconnected
   * providers whose historical invoices still live in the DB.
   */
  availableSources?: string[];
};
```

AuditEventType TYPE

packages\dtos\src\dtos\audit.dto.ts

```
export type AuditEventType = (typeof AUDIT_EVENT_TYPES)[number];
```

AuditLogDTO TYPE

packages\dtos\src\dtos\audit.dto.ts

references: AuditEventType, AuditResourceType

```
/**
 * Backend read model — rows returned from the AuditLogs table.
 * id is a BIGINT IDENTITY, so it is typed as number.
 */
export type AuditLogDTO = {
  id: number;
  businessId: string;
  actorId: string | null;
  actorEmail: string | null;
  actorRole: string | null;
  ipAddress: string | null;
  eventType: AuditEventType;
  resourceType: AuditResourceType | null;
  resourceId: string | null;
  payload: Record<string, unknown>;
  createdAt: number; // epoch ms
};
```

AuditResourceType TYPE

packages\dtos\src\dtos\audit.dto.ts

```
export type AuditResourceType =
  | 'INVOICE'
  | 'INVOICE_PAYMENT'
  | 'USER'
  | 'JOB'
  | 'BATCH'
  | 'CONFIG'
  | 'AUDIT_LOG'
  | 'XERO_CONNECTION'
  | 'SAGE_CONNECTION'
  | 'MYOB_CONNECTION'
  | 'TILL'
  | 'API_KEY';
```

BulkCancelBlockedPayment TYPE

packages\dtos\src\dtos\api\invoices.api.ts

references: BulkCancelBlockReason

```
export type BulkCancelBlockedPayment = {
  paymentId: string;
  reason: BulkCancelBlockReason;
};
```

BulkCancelBlockReason TYPE

packages\dtos\src\dtos\api\invoices.api.ts

```
/**
 * Reasons a single payment can be excluded from a bulk cancellation request.
 * Mirror of the per-payment validation in the single-cancel service, surfaced
 * inline so the client can render a per-row warning without a follow-up call.
 */
export type BulkCancelBlockReason =
  | 'PAYMENT_NOT_FOUND'
  | 'PAYMENT_NOT_CANCELLABLE'
  | 'PAYMENT_ALREADY_CANCELLED'
  | 'INVOICE_TYPE_NOT_SUPPORTED'
  | 'MISSING_FISCAL_REFERENCE';
```

BulkCancelInFlightPayment TYPE

packages\dtos\src\dtos\api\invoices.api.ts

```
/**
 * Surfaced when the payment is already being processed for cancellation
 * (status is `imported` or `pending` with a non-zero `cancellationAttempt`).
 * The client can wait for the current attempt to reach a terminal state.
 */
export type BulkCancelInFlightPayment = {
  paymentId: string;
};
```

BulkCancelInvoiceResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

references: BulkCancelBlockedPayment, BulkCancelInFlightPayment

```
/**
 * Response from POST /api/v1/businesses/:businessId/invoices/cancel-bulk.
 *
 * Fiscalisation jobs are capped at `MAX_PAYMENTS_PER_FISCALISATION_JOB`
 * payments each, so a bulk cancel of more than that count is split into
 * multiple jobs. `jobIds` contains every dispatched job id in the order they
 * were dispatched. Empty `[]` when nothing was dispatched (every input was
 * blocked or in-flight).
 *
 * Poll each id via `useGetFiscalisationJobQuery` (or aggregate via
 * `useAggregatedFiscalisationJobs` on the app side) to surface combined status.
 */
export type BulkCancelInvoiceResponse = {
  jobIds: string[];
  /** The new cancellation payment ids — one per successfully dispatched cancel. */
  cancelledPaymentIds: string[];
  blocked: BulkCancelBlockedPayment[];
  inFlight: BulkCancelInFlightPayment[];
};
```

CancelInvoiceResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

```
/**
 * Response from POST
/api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/cancel.
 * A new cancellation payment row is created on the parent invoice (linked to
 * the original payment via `CancelsPaymentId`) and dispatched for
 * fiscalisation; the original payment transitions to `Status='cancelled'`
 * synchronously. Poll `jobId` until terminal — on success the cancellation
 * payment row reaches `Status='fiscalised'`.
 */
export type CancelInvoiceResponse = {
  jobId: string;
  /** The new payment row created as the cancellation payment. */
  cancellationPaymentId: string;
};
```

CertificateDTO CLASS

packages\dtos\src\dtos\certificate.dto.ts

references: CertificateDTOFields, CertificateRow, CertificateStatus

```
export class CertificateDTO {
  certificateId: string;
  locationId: string;
  businessId: string;
  name: string;
  vsdcCertificateFile: string;
  vsdcPfxPassword: string;
  vsdcPac: string;
  uid: string;
  taxCoreApiUrl: string | null;
  supportedLanguages: string[] | null;
  status: CertificateStatus;
  isDefault: boolean;
  registeredAt: number;
  revokedAt: number | null;
  createdAt: number;
  updatedAt: number;

  constructor(fields: CertificateDTOFields) {
    this.certificateId = fields.certificateId;
    this.locationId = fields.locationId;
    this.businessId = fields.businessId;
    this.name = fields.name;
    this.vsdcCertificateFile = fields.vsdcCertificateFile;
    this.vsdcPfxPassword = fields.vsdcPfxPassword;
    this.vsdcPac = fields.vsdcPac;
    this.uid = fields.uid;
    this.taxCoreApiUrl = fields.taxCoreApiUrl;
    this.supportedLanguages = fields.supportedLanguages;
    this.status = fields.status;
    this.isDefault = fields.isDefault;
    this.registeredAt = fields.registeredAt;
    this.revokedAt = fields.revokedAt;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: CertificateRow): CertificateDTO {
    return new CertificateDTO({
      certificateId: row.CertificateId,
      locationId: row.LocationId,
      businessId: row.BusinessId,
      name: row.Name,
      vsdcCertificateFile: row.VsdcCertificateFile,
      vsdcPfxPassword: row.VsdcPfxPassword,
```

```
      vsdcPac: row.VsdcPac,
      uid: row.Uid,
      taxCoreApiUrl: row.TaxCoreApiUrl ?? null,
      supportedLanguages: parseSupportedLanguages(row.SupportedLanguages),
      status: row.Status,
      isDefault: Boolean(row.IsDefault),
      registeredAt: Number(row.RegisteredAt),
      revokedAt: row.RevokedAt != null ? Number(row.RevokedAt) : null,
      createdAt: Number(row.CreatedAt),
      updatedAt: Number(row.UpdatedAt),
    });
  }
}
```

CertificateDTOFields TYPE

packages\dtos\src\dtos\certificate.dto.ts

references: CertificateStatus

```
export type CertificateDTOFields = {
  certificateId: string;
  locationId: string;
  businessId: string;
  name: string;
  vsdcCertificateFile: string;
  /** Decrypted plaintext. Internal use only — never serialised to API responses. */
  vsdcPfxPassword: string;
  /** Decrypted plaintext. Internal use only — never serialised to API responses. */
  vsdcPac: string;
  uid: string;
  taxCoreApiUrl: string | null;
  supportedLanguages: string[] | null;
  status: CertificateStatus;
  isDefault: boolean;
  registeredAt: number;
  revokedAt: number | null;
  createdAt: number;
  updatedAt: number;
};
```

CertificateRow TYPE

packages\dtos\src\dtos\certificate.dto.ts

references: CertificateStatus

```
export type CertificateRow = {
  CertificateId: string;
  LocationId: string;
  BusinessId: string;
  Name: string;
  VsdcCertificateFile: string;
  /** AES-256-GCM ciphertext. Decrypted by the repository before mapping. */
  VsdcPfxPassword: string;
  /** AES-256-GCM ciphertext. Decrypted by the repository before mapping. */
  VsdcPac: string;
  Uid: string;
  TaxCoreApiUrl: string | null;
  /** JSON-encoded string array of locale codes (e.g. `["en-US"]`). */
  SupportedLanguages: string | null;
  Status: CertificateStatus;
  IsDefault: boolean;
  RegisteredAt: number;
  RevokedAt: number | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

CertificateStatus TYPE

packages\dtos\src\dtos\certificate.dto.ts

```
/**
 * Certificate — one row per issued PFX file under a location. Introduced
 * in migration 042 (TAXCORE-34). VsdcPfxPassword and VsdcPac are stored
 * AES-256-GCM-encrypted on disk; the backend repo decrypts them before
 * mapping `LocationRow → LocationDTO`, so callers always see plaintext.
 * Public API responses strip the password/PAC fields via a controller
 * helper — they are write-only.
 */
export type CertificateStatus = 'active' | 'revoked';
```

ChangePasswordResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

```
export type ChangePasswordResponseDTO = {
  accessToken: string;
  refreshToken: string;
};
```

ConnectorType ENUM

packages\dtos\src\dtos\accounting\connectorType.enum.ts

```
/**
 * Enumerates the supported fiscalisation connectors — the accounting-system
 * integrations plus the generic HTTP connector (TAXCORE-340: renamed in place
 * from the former provider-type enum). Shared-kernel vocabulary: it stays in
 * `packages/dtos` and the backend fiscalisation domain imports it from here.
 * Adding a new connector is a single entry here plus a new adapter.
 */
export enum ConnectorType {
  Xero = 'xero',
  Sage = 'sage',
  Myob = 'myob',
  Sage100 = 'sage100',
  Sage300 = 'sage300',
  Sage200Evolution = 'sage200evolution',
  Amicus = 'amicus',
  Atrex = 'atrex',
  Http = 'http',
}
```

CopyInvoiceResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

references: CopyPaymentResponse

```
/** @deprecated alias kept for legacy callers; identical to CopyPaymentResponse. */
export type CopyInvoiceResponse = CopyPaymentResponse;
```

CopyPaymentResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

```
/**
 * Response from
 * `POST /api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/copy`.
 *
 * Per-payment Copy: a new `Invoices` row is created with
 * `InvoiceType='COPY'` and `CopiesInvoiceId` pointing back at the parent
 * (source) invoice. A single synthetic `InvoicePayments` row is inserted
 * under it carrying `CopiesPaymentId` pointing at the source payment, and
 * `PaymentAmount` equal to the source payment's amount. The Copy V-SDC
 * submission's `referentDocumentNumber` is the source payment's SDC fiscal
 * invoice number.
 *
 * Each source payment can be copied at most once — a second attempt
 * returns `409 INVOICE_ALREADY_COPIED`. The source invoice's other
 * payments can each be copied independently (one Copy invoice per copied
 * payment).
 *
 * `jobId` is the dispatched fiscalisation job for the new copy's
 * synthetic payment, or null when the eligibility validator blocked
 * dispatch (the copy row still exists and can be fiscalised later).
 */
export type CopyPaymentResponse = {
  /** Invoice id of the newly-created copy row (NOT the original). */
  invoiceId: string;
  /** Synthetic payment id under the new copy row. */
  invoicePaymentId: string;
  /** External reference (invoice number) of the copy row. */
  invoiceNumber: string;
  /** Fiscalisation job id, or null when dispatch was blocked at the
   *  eligibility check. */
  jobId: string | null;
};
```

CreateApiKeyResponse TYPE

packages\dtos\src\dtos\apiKey.dto.ts

references: ApiKeyDTO

```
export type CreateApiKeyResponse = ApiKeyDTO & {
  key: string;
};
```

CreateMcpTokenResponse TYPE

packages\dtos\src\dtos\mcp.dto.ts

references: McpTokenDTO

```
export type CreateMcpTokenResponse = McpTokenDTO & {
  token: string;
};
```

DailyFiscalReportPaymentBucket TYPE

packages\dtos\src\dtos\api\dailyFiscalReport.api.ts

```
/**
 * Per-payment-type bucket — every `PaymentType` enum value appears in the
 * response, including zero buckets, so the UI and CSV stay self-documenting
 * for empty days.
 */
export type DailyFiscalReportPaymentBucket = {
  count: number;
  amount: number;
};
```

DailyFiscalReportResponse TYPE

packages\dtos\src\dtos\api\dailyFiscalReport.api.ts

references: DailyFiscalReportPaymentBucket, InvoiceType, PaymentType, TransactionType

```ts
/**
 * Server-side aggregation of fiscalised payments inside the chosen date
 * window. Aggregation is payment-aware: every fiscalised `InvoicePayments`
 * row counts as one fiscal event and amounts come from `PaymentAmount`
 * (not `Invoices.TotalAmount`), so multi-payment invoices split across
 * days correctly.
 *
 * Sections mirror the four bullet points in TAXCORE-160:
 *   1. byInvoiceType           — total payment count per InvoiceType
 *   2. byTransactionType       — total payment count per TransactionType
 *   3. saleRefundTotals        — Normal + Advance only, summed amounts
 *   4. byPaymentType           — per PaymentType { count, amount }
 *
 * All record maps are seeded with every enum value (zero counts where
 * appropriate) before the response is built — callers can index every
 * enum key without a presence check.
 */
export type DailyFiscalReportResponse = {
  /** ISO-8601 — echoed from the request, useful for the CSV header. */
  dateFrom: string;
  dateTo: string;
  byInvoiceType: Record<InvoiceType, number>;
  byTransactionType: Record<TransactionType, number>;
  saleRefundTotals: {
    saleAmount: number;
    refundAmount: number;
  };
  byPaymentType: Record<PaymentType, DailyFiscalReportPaymentBucket>;
};
```

FiscalInvoiceRecordDTO TYPE

packages\dtos\src\dtos\fiscalRecord.dto.ts

```ts
/**
 * TaxCore V-SDC fiscal invoice record returned by
 * GET /api/v3/invoices/{requestId}.
 *
 * `encryptedInternalData` from the raw V-SDC response is intentionally
 * omitted — it contains TaxCore-internal encrypted data and must never
 * be exposed to frontend clients.
 */
export type FiscalInvoiceRecordDTO = {
  /** The 8-char RequestId that was sent as the V-SDC RequestId header. */
  requestId: string;
  /** Fiscal invoice number assigned by the SDC (e.g. "27/2024PB10002VL"). */
  invoiceNumber: string;
  /** SDC invoice counter string (e.g. "27/2024PB10002VL"). */
  invoiceCounter: string;
  /** Verification URL for the fiscal receipt. Null when not returned by V-SDC. */
  verificationUrl: string | null;
  /** QR-code data URL for the fiscal receipt. Null when omitted via options. */
  verificationQRCode: string | null;
  /** Signed hash (digital signature) of the invoice. Null when not returned. */
  signature: string | null;
  /** ISO 8601 date-time string from the SDC clock. */
  sdcDateTime: string;
  /** Total invoice amount as returned by V-SDC. */
  totalAmount: number;
  /** Name / identifier of the SDC that signed the invoice. */
  signedBy: string;
  /** Journal string as returned by V-SDC. Null when not returned. */
  journal: string | null;
};
```

FiscalisationJobDTO CLASS

packages\dtos\src\dtos\fiscalisationJob.dto.ts

references: FiscalisationJobDTOFields, FiscalisationJobResultItem, FiscalisationJobRow

```
export class FiscalisationJobDTO {
  fiscalisationJobId: string;
  businessId: string;
  status: string;
  /**
   * `InvoicePayments.InvoicePaymentId[]` for every payment in this job.
   * Surfaced so the app can render an optimistic Pending status on those
   * rows while the consumer churns through the queue — without needing to
   * reach into `results`, which is empty until payments start completing.
   */
  paymentIds: string[];
  totalInvoices: number;
  fiscalisedCount: number;
  failedCount: number;
  results: FiscalisationJobResultItem[];
  createdAt: number;
  updatedAt: number;

  constructor(fields: FiscalisationJobDTOFields) {
    this.fiscalisationJobId = fields.fiscalisationJobId;
    this.businessId = fields.businessId;
    this.status = fields.status;
    this.paymentIds = fields.paymentIds;
    this.totalInvoices = fields.totalInvoices;
    this.fiscalisedCount = fields.fiscalisedCount;
    this.failedCount = fields.failedCount;
    this.results = fields.results;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: FiscalisationJobRow): FiscalisationJobDTO {
    return new FiscalisationJobDTO({
      fiscalisationJobId: row.FiscalisationJobId,
      businessId: row.BusinessId,
      status: row.Status,
      paymentIds: safeParseJson<string[]>(
        row.PaymentIds,
        [],
        row.FiscalisationJobId,
        'PaymentIds',
      ),
      totalInvoices: row.TotalInvoices,
      fiscalisedCount: row.FiscalisedCount,
      failedCount: row.FailedCount,
```

```
      results: safeParseJson<FiscalisationJobResultItem[]>(
        row.Results,
        [],
        row.FiscalisationJobId,
        'Results',
      ),
      createdAt: row.CreatedAt,
      updatedAt: row.UpdatedAt,
    });
  }
}
```

FiscalisationJobDTOFields TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

references: FiscalisationJobResultItem

```
export type FiscalisationJobDTOFields = {
  fiscalisationJobId: string;
  businessId: string;
  status: string;
  paymentIds: string[];
  totalInvoices: number;
  fiscalisedCount: number;
  failedCount: number;
  results: FiscalisationJobResultItem[];
  createdAt: number;
  updatedAt: number;
};
```

FiscalisationJobResultItem TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

references: FiscalisationJobResultItemError, FiscalisationJobResultItemSuccess

```
/**
 * Per-invoice result from a fiscalisation job. Discriminated over `status`
 * so success-only and error-only fields can't coexist at the type level.
 */
export type FiscalisationJobResultItem =
  | FiscalisationJobResultItemSuccess
  | FiscalisationJobResultItemError;
```

FiscalisationJobResultItemBase TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

```
/**
 * Common fields on every per-payment fiscalisation result. Each item is one
 * V-SDC submission (one `InvoicePayments` row). `invoiceId` is carried so the
 * UI can correlate the result back to its parent invoice without an extra
 * lookup; `invoiceNumber` mirrors the operator-facing reference.
 */
type FiscalisationJobResultItemBase = {
  /** `InvoicePayments.InvoicePaymentId` — one fiscal event per result item. */
  paymentId: string;
  /** Parent invoice — for navigation/correlation. */
  invoiceId: string;
  invoiceNumber: string;
  attempts: number;
};
```

FiscalisationJobResultItemError TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

references: FiscalisationJobResultItemBase

```
export type FiscalisationJobResultItemError = FiscalisationJobResultItemBase & {
  status: 'error';
  errorMessage: string;
};
```

FiscalisationJobResultItemSuccess TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

references: FiscalisationJobResultItemBase

```
export type FiscalisationJobResultItemSuccess =
  FiscalisationJobResultItemBase & {
    status: 'fiscalised';
    fiscalInvoiceNumber: string;
    fiscalTimestamp: number;
    fiscalVerificationUrl?: string;
    /** The 8-char uppercase alphanumeric RequestId sent to V-SDC. */
    fiscalRequestId: string;
    /** Raw V-SDC journal text — null when V-SDC omits textual representation. */
    fiscalJournal?: string | null;
  };
```

FiscalisationJobRow TYPE

packages\dtos\src\dtos\fiscalisationJob.dto.ts

```
export type FiscalisationJobRow = {
  FiscalisationJobId: string;
  BusinessId: string;
  /** JSON-serialised `InvoicePayments.InvoicePaymentId[]` — one fiscal event per id. */
  PaymentIds: string;
  /** Count of payments (fiscal events) — keeps the legacy column name for backwards
compat. */
  TotalInvoices: number;
  FiscalisedCount: number;
  FailedCount: number;
  Status: string;
  Results: string | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

ForgotPasswordResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

```
export type ForgotPasswordResponseDTO = {
  message: string;
};
```

GetInvoicePaymentByIdResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

references: InvoicePaymentDTO

```
/**
 * Response from
 * `GET /businesses/:businessId/invoices/:invoiceId/payments/:paymentId`.
 * Returns the single payment. `404 INVOICE_NOT_FOUND` when the payment
 * doesn't belong to the invoice or business (uses the existing error code
 * to keep the response surface flat for clients).
 */
export type GetInvoicePaymentByIdResponse = InvoicePaymentDTO;
```

GetInvoicePaymentsResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

references: InvoicePaymentDTO

```
/**
 * Response from `GET /businesses/:businessId/invoices/:invoiceId/payments`.
 * Returns every `InvoicePayments` row under the invoice in
 * `(PaymentDate ASC, CreatedAt ASC)` order — same as the inline `payments[]`
 * array on `GetInvoiceByIdResponse`. The standalone endpoint exists so the
 * FE can drill into the payments list without re-fetching the parent
 * invoice when only the payments are needed (refresh after a per-payment
 * action, deep-link to a specific payment, etc.).
 */
export type GetInvoicePaymentsResponse = {
  payments: InvoicePaymentDTO[];
};
```

ImportBatchDTO CLASS

packages\dtos\src\dtos\importBatch.dto.ts

references: ImportBatchDTOFields, ImportBatchRow, ImportRowError

```
export class ImportBatchDTO {
  importBatchId: string;
  businessId: string;
  source: string;
  filename: string;
  totalRows: number;
  importedRows: number;
  failedRows: number;
  status: string;
  errors?: ImportRowError[];
  createdAt: number;
  updatedAt: number;

  constructor(fields: ImportBatchDTOFields) {
    this.importBatchId = fields.importBatchId;
    this.businessId = fields.businessId;
    this.source = fields.source;
    this.filename = fields.filename;
    this.totalRows = fields.totalRows;
    this.importedRows = fields.importedRows;
    this.failedRows = fields.failedRows;
    this.status = fields.status;
    this.errors = fields.errors;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: ImportBatchRow, errors?: ImportRowError[]): ImportBatchDTO {
    return new ImportBatchDTO({
      importBatchId: row.ImportBatchId,
      businessId: row.BusinessId,
      source: row.Source,
      filename: row.Filename,
      totalRows: row.TotalRows,
      importedRows: row.ImportedRows,
      failedRows: row.FailedRows,
      status: row.Status,
      errors,
      createdAt: row.CreatedAt,
      updatedAt: row.UpdatedAt,
    });
  }
}
```

ImportBatchDTOFields TYPE

packages\dtos\src\dtos\importBatch.dto.ts

references: ImportRowError

```
export type ImportBatchDTOFields = {
  importBatchId: string;
  businessId: string;
  source: string;
  filename: string;
  totalRows: number;
  importedRows: number;
  failedRows: number;
  status: string;
  errors?: ImportRowError[];
  createdAt: number;
  updatedAt: number;
};
```

ImportBatchRow TYPE

packages\dtos\src\dtos\importBatch.dto.ts

```
export type ImportBatchRow = {
  ImportBatchId: string;
  BusinessId: string;
  Source: string;
  Filename: string;
  TotalRows: number;
  ImportedRows: number;
  FailedRows: number;
  Status: string;
  CreatedAt: number;
  UpdatedAt: number;
};
```

ImportRowError TYPE

packages\dtos\src\dtos\importBatch.dto.ts

```
export type ImportRowError = {
  row: number;
  invoiceNumber: string;
  reason: string;
};
```

InviteDTO TYPE

packages\dtos\src\dtos\invite.dto.ts

references: UserRole

```
export type InviteDTO = {
  inviteToken: string;
  email: string;
  roles: UserRole[];
  organizationId: string;
  inviteUrl: string;
};
```

InvoiceAggregateStatus TYPE

packages\dtos\src\dtos\invoice.dto.ts

```
/**
 * Aggregate status for a multi-payment invoice. Derived in
 * `InvoiceDTO.toDto` from the statuses of `payments[]`:
 *   * `imported`   — all payments imported (none fiscalised yet)
 *   * `partial`    — at least one fiscalised, at least one not yet
 *   * `fiscalised` — all payments fiscalised (or cancelled — terminal-OK)
 *   * `cancelled`  — every payment cancelled, none active
 *   * `error`      — any payment terminally errored
 */
export type InvoiceAggregateStatus =
  | 'imported'
  | 'partial'
  | 'fiscalised'
  | 'cancelled'
  | 'error';
```

InvoiceDTO CLASS

packages\dtos\src\dtos\invoice.dto.ts

references: InvoiceAggregateStatus, InvoiceDTOFields, InvoiceLineItemDTO, InvoicePaymentDTO, InvoiceRow, InvoiceType, NormalizedInvoice, PaymentType, TransactionType

```
export class InvoiceDTO {
  invoiceId: string;
  businessId: string;
  importBatchId: string;
  invoiceNumber: string;
  reference: string | null;
  source: string;
  /**
   * Denormalised aggregate status persisted on `Invoices.Status` — kept for
   * cheap filtering. The authoritative value is `aggregateStatus`, computed
   * from `payments[]` at read time.
   */
  status: string;
  /**
   * Derived from `payments[]`. Surfaced to the UI as the headline status badge.
   */
  aggregateStatus: InvoiceAggregateStatus;
  invoiceDate: number;
  dueDate: number | null;
  buyerName: string | null;
  buyerTin: string | null;
  buyerAddress: string | null;
  /** Buyer email address (TAXCORE-308). Null until set. */
  buyerEmail: string | null;
  /** Buyer cost-centre identifier — sent to V-SDC. Null until set. */
  buyerCostCenterId: string | null;
  /** Cashier / operator identifier sent to V-SDC as the `cashier` field. Null maps to
default '1' in the mapper. */
  cashierId: string | null;
  /** Reference to the original fiscalised document (corrective / refund invoices). */
  referentDocumentNumber: string | null;
  /** ISO 8601 date-time of the referent document (corrective / refund invoices). */
  referentDocumentDT: string | null;
  currencyCode: string;
  subtotalAmount: number;
  taxAmount: number;
  totalAmount: number;
  /** V-SDC InvoiceType (NORMAL / PROFORMA / COPY / TRAINING / ADVANCE). Null = default
NORMAL. */
  invoiceType: InvoiceType | null;
  /** V-SDC TransactionType (SALE / REFUND). Null = derive from totalAmount sign. */
  transactionType: TransactionType | null;
  /** V-SDC PaymentType (CASH / CARD / OTHER). Null = default CASH. */
  paymentType: PaymentType | null;
```

```
  /** Arbitrary key-value pairs sent as InvoiceAdditionalFields to V-SDC. */
  invoiceAdditionalFields: Record<string, string> | null;
  /** Arbitrary key-value pairs sent as PaymentAdditionalFields inside the payment array.
*/
  paymentAdditionalFields: Record<string, string> | null;
  /**
   * Internal breadcrumbs stamped by the normaliser at sync time (provider IDs,
   * source status, etc.). Never forwarded to V-SDC.
   */
  rawSourceData: Record<string, string> | null;
  lineItems: InvoiceLineItemDTO[];
  /**
   * One entry per fiscal event. Drives the per-payment receipt UI and the
   * per-payment fiscalise / cancel actions. Empty array is valid — e.g. an
   * AUTHORISED Xero invoice with no payments recorded yet.
   */
  payments: InvoicePaymentDTO[];
  /**
   * Set on a COPY invoice row; FK to the original fiscalised invoice this row
   * is a Copy Sale / Copy Refund of. NULL on regular (non-copy) invoices.
   * Used by the frontend to recognise a Copy detail screen and restrict its
   * available actions to Close only.
   */
  copiesInvoiceId: string | null;
  /**
   * Set on a REFUND invoice row created via the per-payment Refund action;
   * FK to the original SALE invoice. NULL on everything else. Frontends
   * treat a refund invoice the same way they treat a copy — no further
   * fiscal actions are available (re-refunding a refund is meaningless;
   * each refund row is itself fiscalised exactly once).
   */
  refundsInvoiceId: string | null;
  /**
   * Epoch ms when the source document was voided in the upstream accounting
   * system (Xero VOIDED today). NULL = active. Set only on SALE invoices;
   * the Refund invoices created in response are not themselves voided.
   */
  voidedAt: number | null;
  /**
   * Set on a regular invoice that sync linked back to a Proforma (quote)
   * source row. NULL on quotes and on invoices without a matching quote.
   */
  convertedFromQuoteId: string | null;
  /**
   * Epoch ms the quote was marked converted. Stamped on the source quote
   * row when sync resolves a regular invoice's `reference` to it. NULL on
   * unconverted quotes and on regular invoices.
   */
  convertedAt: number | null;
  /**
   * Reverse navigation — set on a PROFORMA (quote) row when sync has
```

```
   * already linked an invoice to it. Populated via reverse lookup on
   * `Invoices.ConvertedFromQuoteId`; lets the frontend show "View Invoice"
   * on the quote detail screen. NULL on regular invoices and on
   * unconverted quotes.
   */
  convertedToInvoiceId: string | null;
  /**
   * Sibling rows that reference this invoice — i.e. Copy invoices
   * (`CopiesInvoiceId = this.invoiceId`) and Refund invoices
   * (`RefundsInvoiceId = this.invoiceId`). Populated by the detail-view
   * repo path (`findById`); empty `[]` in list/sync contexts and on the
   * linked rows themselves (no recursion). Surfaced in the FE as a
   * "Linked Invoices" section under Payments so an admin can see the
   * Refund created in response to a Xero void (or a manual Copy/Refund)
   * without leaving the source invoice's detail screen.
   */
  linkedInvoices: InvoiceDTO[];
  /**
   * Captures which secure-element certificate signed this invoice
   * (migration 042, TAXCORE-34). NULL means the fiscalisation worker will
   * resolve the business's default certificate at dispatch time.
   */
  certificateId: string | null;
  /**
   * The Location this invoice was fiscalised under (TAXCORE-246). Stamped from
   * the pushing API key's location; NULL on legacy / OAuth-sync rows.
   */
  locationId: string | null;
  /**
   * The POS till this invoice was routed through (TAXCORE-285). Stamped from
   * the AMICUS / Sage 200 Evolution register → till routing; NULL on legacy /
   * OAuth-sync / CSV rows. Drives the Home dashboard's POS Till filter.
   */
  tillId: string | null;
  /**
   * AMICUS posSaleId (TAXCORE-293). For a refund row, carries the
   * composite `<sale>:reverse:<paymentId>` for a per-payment reversal and
   * the bare ReverseSale-id for a whole-sale void — the FE uses the
   * `:reverse:` substring to partition refunds into REVERSE SALES vs
   * REVERSE PAYMENTS sections. NULL on every non-AMICUS row.
   */
  posSaleId: string | null;
  createdAt: number;
  updatedAt: number;

  constructor(fields: InvoiceDTOFields) {
    this.invoiceId = fields.invoiceId;
    this.businessId = fields.businessId;
    this.importBatchId = fields.importBatchId;
    this.invoiceNumber = fields.invoiceNumber;
    this.reference = fields.reference;
```

```
    this.source = fields.source;
    this.status = fields.status;
    this.aggregateStatus = fields.aggregateStatus;
    this.invoiceDate = fields.invoiceDate;
    this.dueDate = fields.dueDate;
    this.buyerName = fields.buyerName;
    this.buyerTin = fields.buyerTin;
    this.buyerAddress = fields.buyerAddress;
    this.buyerEmail = fields.buyerEmail;
    this.buyerCostCenterId = fields.buyerCostCenterId;
    this.cashierId = fields.cashierId;
    this.referentDocumentNumber = fields.referentDocumentNumber;
    this.referentDocumentDT = fields.referentDocumentDT;
    this.currencyCode = fields.currencyCode;
    this.subtotalAmount = fields.subtotalAmount;
    this.taxAmount = fields.taxAmount;
    this.totalAmount = fields.totalAmount;
    this.invoiceType = fields.invoiceType;
    this.transactionType = fields.transactionType;
    this.paymentType = fields.paymentType;
    this.invoiceAdditionalFields = fields.invoiceAdditionalFields;
    this.paymentAdditionalFields = fields.paymentAdditionalFields;
    this.rawSourceData = fields.rawSourceData;
    this.lineItems = fields.lineItems;
    this.payments = fields.payments;
    this.copiesInvoiceId = fields.copiesInvoiceId;
    this.refundsInvoiceId = fields.refundsInvoiceId;
    this.voidedAt = fields.voidedAt;
    this.convertedFromQuoteId = fields.convertedFromQuoteId;
    this.convertedAt = fields.convertedAt;
    this.convertedToInvoiceId = fields.convertedToInvoiceId;
    this.linkedInvoices = fields.linkedInvoices;
    this.certificateId = fields.certificateId;
    this.locationId = fields.locationId;
    this.tillId = fields.tillId;
    this.posSaleId = fields.posSaleId;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(
    row: InvoiceRow,
    lineItems: InvoiceLineItemDTO[] = [],
    payments: InvoicePaymentDTO[] = [],
    convertedToInvoiceId: string | null = null,
    linkedInvoices: InvoiceDTO[] = [],
  ): InvoiceDTO {
    return new InvoiceDTO({
      invoiceId: row.InvoiceId,
      businessId: row.BusinessId,
      importBatchId: row.ImportBatchId,
```

```
      invoiceNumber: row.ExternalReference,
      // Surface the provider's `reference` field so the FE can show
      // "Converted from quote {reference}" copy. Previously hard-coded to
      // null while only RawSourceData carried the value.
      reference: row.Reference ?? null,
      source: row.Source,
      status: row.Status,
      aggregateStatus: deriveAggregateStatus(payments, row.Status),
      invoiceDate: row.InvoiceDate,
      dueDate: row.DueDate,
      buyerName: row.BuyerName,
      buyerTin: row.BuyerTin,
      buyerAddress: row.BuyerAddress,
      buyerEmail: row.BuyerEmail ?? null,
      buyerCostCenterId: row.BuyerCostCenterId,
      cashierId: row.CashierId ?? null,
      referentDocumentNumber: row.ReferentDocumentNumber,
      referentDocumentDT: row.ReferentDocumentDT,
      currencyCode: row.CurrencyCode,
      subtotalAmount: row.SubtotalAmount,
      taxAmount: row.TaxAmount,
      totalAmount: row.TotalAmount,
      invoiceType: row.InvoiceType as InvoiceType | null,
      transactionType: row.TransactionType as TransactionType | null,
      paymentType: row.PaymentType as PaymentType | null,
      invoiceAdditionalFields: row.InvoiceAdditionalFields
        ? JSON.parse(row.InvoiceAdditionalFields)
        : null,
      paymentAdditionalFields: row.PaymentAdditionalFields
        ? JSON.parse(row.PaymentAdditionalFields)
        : null,
      rawSourceData: row.RawSourceData
        ? (JSON.parse(row.RawSourceData) as Record<string, string>)
        : null,
      lineItems,
      payments,
      copiesInvoiceId: row.CopiesInvoiceId ?? null,
      refundsInvoiceId: row.RefundsInvoiceId ?? null,
      voidedAt: row.VoidedAt != null ? Number(row.VoidedAt) : null,
      convertedFromQuoteId: row.ConvertedFromQuoteId ?? null,
      convertedAt: row.ConvertedAt != null ? Number(row.ConvertedAt) : null,
      convertedToInvoiceId,
      linkedInvoices,
      certificateId: row.CertificateId ?? null,
      locationId: row.LocationId ?? null,
      tillId: row.TillId ?? null,
      posSaleId: row.PosSaleId ?? null,
      createdAt: row.CreatedAt,
      updatedAt: row.UpdatedAt,
    });
  }
```

```
/**
 * Builds a local (pre-upload) InvoiceDTO from a NormalizedInvoice.
 *
 * The resulting DTO has a synthetic `invoiceId` prefixed with "local-" and
 * an empty payments array — the import path attaches synthetic payment
 * rows server-side after the parent insert.
 */
static fromNormalized(
  invoice: NormalizedInvoice,
  batchId: string,
  businessId: string,
): InvoiceDTO {
  const now = Date.now();
  const invoiceId = `local-${invoice.idempotencyKey}`;
  return new InvoiceDTO({
    invoiceId,
    businessId,
    importBatchId: batchId,
    invoiceNumber: invoice.invoiceNumber,
    reference: invoice.reference ?? null,
    source: invoice.source,
    status: InvoiceStatus.Imported,
    aggregateStatus: 'imported',
    invoiceDate: invoice.invoiceDate,
    dueDate: invoice.dueDate ?? null,
    buyerName: invoice.buyerName ?? null,
    buyerTin: invoice.buyerTin ?? null,
    buyerAddress: invoice.buyerAddress ?? null,
    buyerEmail: invoice.buyerEmail ?? null,
    buyerCostCenterId: invoice.buyerCostCenterId ?? null,
    cashierId: invoice.cashierId ?? null,
    referentDocumentNumber: invoice.referentDocumentNumber ?? null,
    referentDocumentDT: invoice.referentDocumentDT ?? null,
    currencyCode: invoice.currencyCode,
    subtotalAmount: invoice.subtotalAmount,
    taxAmount: invoice.taxAmount,
    totalAmount: invoice.totalAmount,
    invoiceType: invoice.invoiceType ?? null,
    transactionType: invoice.transactionType ?? null,
    paymentType: invoice.paymentType ?? null,
    invoiceAdditionalFields: invoice.invoiceAdditionalFields ?? null,
    paymentAdditionalFields: invoice.paymentAdditionalFields ?? null,
    rawSourceData: invoice.rawSourceData ?? null,
    payments: invoice.payments.map((p) =>
      InvoicePaymentDTO.fromNormalized(p, invoiceId, businessId),
    ),
    copiesInvoiceId: null,
    refundsInvoiceId: null,
    voidedAt: null,
    convertedFromQuoteId: null,
```

```
      convertedAt: null,
      convertedToInvoiceId: null,
      linkedInvoices: [],
      certificateId: null,
      locationId: null,
      tillId: null,
      posSaleId: null,
      createdAt: now,
      updatedAt: now,
      lineItems: invoice.lineItems.map(
        (item, idx) =>
          new InvoiceLineItemDTO({
            invoiceLineItemId: `local-${invoice.idempotencyKey}-${idx}`,
            invoiceId,
            businessId,
            description: item.description,
            gtin: item.gtin ?? null,
            quantity: item.quantity,
            unitPrice: item.unitPrice,
            taxRatePercent: item.taxRatePercent,
            taxLabel: item.taxLabel,
            lineSubtotal: item.lineSubtotal,
            lineTaxAmount: item.lineTaxAmount,
            lineTotal: item.lineTotal,
            sortOrder: item.sortOrder,
            itemAdditionalFields: item.itemAdditionalFields ?? null,
            createdAt: now,
            updatedAt: now,
          }),
      ),
    });
  }
}
```

InvoiceDTOFields TYPE

packages\dtos\src\dtos\invoice.dto.ts

references: InvoiceAggregateStatus, InvoiceDTO, InvoiceLineItemDTO, InvoicePaymentDTO, InvoiceType, PaymentType, TransactionType

```
export type InvoiceDTOFields = {
  invoiceId: string;
  businessId: string;
  importBatchId: string;
  invoiceNumber: string;
  reference: string | null;
  source: string;
  status: string;
  aggregateStatus: InvoiceAggregateStatus;
  invoiceDate: number;
  dueDate: number | null;
  buyerName: string | null;
  buyerTin: string | null;
  buyerAddress: string | null;
  /** Buyer email address (TAXCORE-308). Null when not supplied. */
  buyerEmail: string | null;
  buyerCostCenterId: string | null;
  cashierId: string | null;
  referentDocumentNumber: string | null;
  referentDocumentDT: string | null;
  currencyCode: string;
  subtotalAmount: number;
  taxAmount: number;
  totalAmount: number;
  invoiceType: InvoiceType | null;
  transactionType: TransactionType | null;
  paymentType: PaymentType | null;
  invoiceAdditionalFields: Record<string, string> | null;
  paymentAdditionalFields: Record<string, string> | null;
  rawSourceData: Record<string, string> | null;
  lineItems: InvoiceLineItemDTO[];
  payments: InvoicePaymentDTO[];
  copiesInvoiceId: string | null;
  refundsInvoiceId: string | null;
  voidedAt: number | null;
  convertedFromQuoteId: string | null;
  convertedAt: number | null;
  convertedToInvoiceId: string | null;
  linkedInvoices: InvoiceDTO[];
  /** The certificate (and by extension, location) used to sign this invoice. */
  certificateId: string | null;
  /** The Location this invoice was fiscalised under (TAXCORE-246). */
  locationId: string | null;
  /** The POS till this invoice was routed through (TAXCORE-285). */
```

```
  tillId: string | null;
  /**
   * AMICUS posSaleId surfaced on the DTO so the FE can partition refunds
   * into REVERSE SALES vs REVERSE PAYMENTS by inspecting the `:reverse:`
   * substring (TAXCORE-293).
   */
  posSaleId: string | null;
  createdAt: number;
  updatedAt: number;
};
```

InvoiceLineItemDTO CLASS

packages\dtos\src\dtos\invoiceLineItem.dto.ts

references: InvoiceLineItemDTOFields, InvoiceLineItemRow, NormalizedLineItem

```
export class InvoiceLineItemDTO implements NormalizedLineItem {
  invoiceLineItemId: string;
  invoiceId: string;
  businessId: string;
  description: string;
  /** Global Trade Item Number (barcode / product code). Null until set. */
  gtin: string | null;
  quantity: number;
  unitPrice: number;
  taxRatePercent: number;
  taxLabel: string | null;
  lineSubtotal: number;
  lineTaxAmount: number;
  lineTotal: number;
  sortOrder: number;
  /** Arbitrary key-value pairs sent as ItemAdditionalFields to V-SDC. */
  itemAdditionalFields: Record<string, string> | null;
  createdAt: number;
  updatedAt: number;

  constructor(fields: InvoiceLineItemDTOFields) {
    this.invoiceLineItemId = fields.invoiceLineItemId;
    this.invoiceId = fields.invoiceId;
    this.businessId = fields.businessId;
    this.description = fields.description;
    this.gtin = fields.gtin;
    this.quantity = fields.quantity;
    this.unitPrice = fields.unitPrice;
    this.taxRatePercent = fields.taxRatePercent;
    this.taxLabel = fields.taxLabel;
    this.lineSubtotal = fields.lineSubtotal;
    this.lineTaxAmount = fields.lineTaxAmount;
    this.lineTotal = fields.lineTotal;
    this.sortOrder = fields.sortOrder;
    this.itemAdditionalFields = fields.itemAdditionalFields;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: InvoiceLineItemRow): InvoiceLineItemDTO {
    return new InvoiceLineItemDTO({
      invoiceLineItemId: row.InvoiceLineItemId,
      invoiceId: row.InvoiceId,
      businessId: row.BusinessId,
      description: row.Description,
```

```
      gtin: row.Gtin,
      quantity: row.Quantity,
      unitPrice: row.UnitPrice,
      taxRatePercent: row.TaxRatePercent,
      taxLabel: row.TaxLabel,
      lineSubtotal: row.LineSubtotal,
      lineTaxAmount: row.LineTaxAmount,
      lineTotal: row.LineTotal,
      sortOrder: row.SortOrder,
      itemAdditionalFields: row.ItemAdditionalFields
        ? JSON.parse(row.ItemAdditionalFields)
        : null,
      createdAt: row.CreatedAt,
      updatedAt: row.UpdatedAt,
    });
  }
}
```

InvoiceLineItemDTOFields `TYPE`

packages\dtos\src\dtos\invoiceLineItem.dto.ts

```
export type InvoiceLineItemDTOFields = {
  invoiceLineItemId: string;
  invoiceId: string;
  businessId: string;
  description: string;
  gtin: string | null;
  quantity: number;
  unitPrice: number;
  taxRatePercent: number;
  taxLabel: string | null;
  lineSubtotal: number;
  lineTaxAmount: number;
  lineTotal: number;
  sortOrder: number;
  itemAdditionalFields: Record<string, string> | null;
  createdAt: number;
  updatedAt: number;
};
```

InvoiceLineItemRow TYPE

packages\dtos\src\dtos\invoiceLineItem.dto.ts

```
export type InvoiceLineItemRow = {
  InvoiceLineItemId: string;
  InvoiceId: string;
  BusinessId: string;
  Description: string;
  Gtin: string | null;
  Quantity: number;
  UnitPrice: number;
  TaxRatePercent: number;
  TaxLabel: string | null;
  LineSubtotal: number;
  LineTaxAmount: number;
  LineTotal: number;
  SortOrder: number;
  /** JSON-serialised Record<string, string> sent as ItemAdditionalFields. Null when not
set. */
  ItemAdditionalFields: string | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

InvoicePaymentDTO CLASS

packages\dtos\src\dtos\invoicePayment.dto.ts

references: InvoicePaymentDTOFields, InvoicePaymentRow, InvoiceStatus, NormalizedInvoicePayment, NormalizedInvoiceTender, PaymentType

```
/**
 * App-facing payment DTO. One per fiscal event. The fiscal fields are
 * populated by the consumer after a successful V-SDC submission; until then
 * they are null.
 */
export class InvoicePaymentDTO {
  invoicePaymentId: string;
  invoiceId: string;
  businessId: string;
  externalPaymentId: string | null;
  paymentAmount: number;
  paymentDate: number;
  /** V-SDC PaymentType resolved at normalisation time. Null when unknown. */
  paymentType: PaymentType | null;
  /**
   * Split-tender breakdown (AMICUS multi-tender — TAXCORE-243). Parsed from
   * the `TenderBreakdown` JSON column. Undefined when NULL or on parse error.
   */
  tenders?: NormalizedInvoiceTender[];
  status: InvoiceStatus;
  fiscalInvoiceNumber: string | null;
  fiscalQrCode: string | null;
  fiscalTimestamp: number | null;
  fiscalVerificationUrl: string | null;
  fiscalSignedHash: string | null;
  fiscalRequestId: string | null;
  fiscalJournal: string | null;
  errorMessage: string | null;
  eligibleForFiscalisation: boolean | null;
  /** Parsed from the CSV column on read. Empty array when none. */
  fiscalisationBlockReasons: string[];
  cancellationAttempt: number;
  cancelsPaymentId: string | null;
  /**
   * Set on the synthetic payment of a COPY invoice; FK to the source
   * payment that was copied.
   */
  copiesPaymentId: string | null;
  /**
   * Set on the synthetic payment of a REFUND invoice; FK to the source
   * SALE payment that was refunded.
   */
  refundsPaymentId: string | null;
  /**
```

```
   * On the SOURCE side of a copy pair, holds the COPY invoice's `invoiceId`
   * so the frontend can navigate from "View Copy" on the source payment row
   * to the copy's detail screen. Populated by the repository at read time
   * via a LEFT JOIN against `InvoicePayments.CopiesPaymentId`. Null when
   * the payment has not been copied (or only an errored copy exists —
   * mirrors `findCopyOf`'s one-time enforcement filter).
   */
  linkedCopyInvoiceId: string | null;
  /**
   * On the SOURCE side of a refund pair, holds the `invoiceId` of EVERY
   * non-errored REFUND invoice that references this source SALE payment, so
   * the frontend can navigate from "View Refund" on the source payment row
   * to each refund's detail screen. Populated by the repository at read time
   * via a LEFT JOIN against `InvoicePayments.RefundsPaymentId` aggregated
   * into an array. Empty array when the payment has not been refunded (or
   * only errored refunds exist — mirrors `findRefundOf`'s one-time
   * enforcement filter). AMICUS sources permit partial reversals, so a
   * single source payment can carry multiple refund links here; Xero / Sage
   * / MYOB sources carry at most one.
   */
  linkedRefundInvoiceIds: string[];
  createdAt: number;
  updatedAt: number;

  constructor(fields: InvoicePaymentDTOFields) {
    this.invoicePaymentId = fields.invoicePaymentId;
    this.invoiceId = fields.invoiceId;
    this.businessId = fields.businessId;
    this.externalPaymentId = fields.externalPaymentId;
    this.paymentAmount = fields.paymentAmount;
    this.paymentDate = fields.paymentDate;
    this.paymentType = fields.paymentType;
    this.tenders = fields.tenders;
    this.status = fields.status;
    this.fiscalInvoiceNumber = fields.fiscalInvoiceNumber;
    this.fiscalQrCode = fields.fiscalQrCode;
    this.fiscalTimestamp = fields.fiscalTimestamp;
    this.fiscalVerificationUrl = fields.fiscalVerificationUrl;
    this.fiscalSignedHash = fields.fiscalSignedHash;
    this.fiscalRequestId = fields.fiscalRequestId;
    this.fiscalJournal = fields.fiscalJournal;
    this.errorMessage = fields.errorMessage;
    this.eligibleForFiscalisation = fields.eligibleForFiscalisation;
    this.fiscalisationBlockReasons = fields.fiscalisationBlockReasons;
    this.cancellationAttempt = fields.cancellationAttempt;
    this.cancelsPaymentId = fields.cancelsPaymentId;
    this.copiesPaymentId = fields.copiesPaymentId;
    this.refundsPaymentId = fields.refundsPaymentId;
    this.linkedCopyInvoiceId = fields.linkedCopyInvoiceId;
    this.linkedRefundInvoiceIds = fields.linkedRefundInvoiceIds;
    this.createdAt = fields.createdAt;
```

```
  this.updatedAt = fields.updatedAt;
}

static toDto(
  row: InvoicePaymentRow,
  linkedCopyInvoiceId: string | null = null,
  linkedRefundInvoiceIds: string[] = [],
): InvoicePaymentDTO {
  return new InvoicePaymentDTO({
    invoicePaymentId: row.InvoicePaymentId,
    invoiceId: row.InvoiceId,
    businessId: row.BusinessId,
    externalPaymentId: row.ExternalPaymentId,
    paymentAmount: row.PaymentAmount,
    paymentDate: row.PaymentDate,
    paymentType: row.PaymentType ?? null,
    tenders: parseTenderBreakdown(row.TenderBreakdown, row.InvoicePaymentId),
    status: row.Status as InvoiceStatus,
    fiscalInvoiceNumber: row.FiscalInvoiceNumber,
    fiscalQrCode: row.FiscalQrCode,
    fiscalTimestamp: row.FiscalTimestamp,
    fiscalVerificationUrl: row.FiscalVerificationUrl,
    fiscalSignedHash: row.FiscalSignedHash,
    fiscalRequestId: row.FiscalRequestId,
    fiscalJournal: row.FiscalJournal,
    errorMessage: row.ErrorMessage,
    eligibleForFiscalisation: row.EligibleForFiscalisation,
    fiscalisationBlockReasons: row.FiscalisationBlockReasons
      ? row.FiscalisationBlockReasons.split(',')
          .map((s) => s.trim())
          .filter(Boolean)
      : [],
    cancellationAttempt: row.CancellationAttempt ?? 0,
    cancelsPaymentId: row.CancelsPaymentId,
    copiesPaymentId: row.CopiesPaymentId ?? null,
    refundsPaymentId: row.RefundsPaymentId ?? null,
    linkedCopyInvoiceId,
    linkedRefundInvoiceIds,
    createdAt: row.CreatedAt,
    updatedAt: row.UpdatedAt,
  });
}

/**
 * Build an in-memory DTO from a normalised payment before the row has hit
 * the DB — used by the CSV import optimistic-UI path so the FE can show a
 * pending payment row without waiting on a round trip. The synthetic
 * `invoicePaymentId` is prefixed `local-` to make it obvious downstream.
 */
static fromNormalized(
  payment: NormalizedInvoicePayment,
```

```
    invoiceId: string,
    businessId: string,
  ): InvoicePaymentDTO {
    const now = Date.now();
    return new InvoicePaymentDTO({
      invoicePaymentId: `local-${payment.idempotencyKey ||
`${invoiceId}:${payment.paymentDate}`}`,
      invoiceId,
      businessId,
      externalPaymentId: payment.externalPaymentId,
      paymentAmount: payment.paymentAmount,
      paymentDate: payment.paymentDate,
      paymentType: payment.paymentType ?? null,
      tenders:
        payment.tenders && payment.tenders.length > 0
          ? payment.tenders
          : undefined,
      status: InvoiceStatus.Imported,
      fiscalInvoiceNumber: null,
      fiscalQrCode: null,
      fiscalTimestamp: null,
      fiscalVerificationUrl: null,
      fiscalSignedHash: null,
      fiscalRequestId: null,
      fiscalJournal: null,
      errorMessage: null,
      eligibleForFiscalisation: payment.eligibleForFiscalisation ?? null,
      fiscalisationBlockReasons: payment.fiscalisationBlockReasons
        ? payment.fiscalisationBlockReasons
            .split(',')
            .map((s) => s.trim())
            .filter(Boolean)
        : [],
      cancellationAttempt: 0,
      cancelsPaymentId: null,
      copiesPaymentId: null,
      refundsPaymentId: null,
      linkedCopyInvoiceId: null,
      linkedRefundInvoiceIds: [],
      createdAt: now,
      updatedAt: now,
    });
  }
}
```

InvoicePaymentDTOFields TYPE

packages\dtos\src\dtos\invoicePayment.dto.ts

references: InvoiceStatus, NormalizedInvoiceTender, PaymentType

```
export type InvoicePaymentDTOFields = {
  invoicePaymentId: string;
  invoiceId: string;
  businessId: string;
  externalPaymentId: string | null;
  paymentAmount: number;
  paymentDate: number;
  /**
   * V-SDC PaymentType resolved for this payment. Null when the provider did
   * not surface enough information (synthetic rows from CSV / Sage / MYOB,
   * or Xero payments whose Account enrichment failed). The consumer's
   * fiscal-request mapper prefers this over `invoice.paymentType`.
   */
  paymentType: PaymentType | null;
  /**
   * Split-tender breakdown (AMICUS multi-tender — TAXCORE-243). Parsed from
   * `InvoicePayments.TenderBreakdown` JSON. Undefined when NULL / parse error.
   */
  tenders?: NormalizedInvoiceTender[];
  status: InvoiceStatus;
  fiscalInvoiceNumber: string | null;
  fiscalQrCode: string | null;
  fiscalTimestamp: number | null;
  fiscalVerificationUrl: string | null;
  fiscalSignedHash: string | null;
  fiscalRequestId: string | null;
  fiscalJournal: string | null;
  errorMessage: string | null;
  eligibleForFiscalisation: boolean | null;
  fiscalisationBlockReasons: string[];
  cancellationAttempt: number;
  cancelsPaymentId: string | null;
  copiesPaymentId: string | null;
  refundsPaymentId: string | null;
  linkedCopyInvoiceId: string | null;
  linkedRefundInvoiceIds: string[];
  createdAt: number;
  updatedAt: number;
};
```

InvoicePaymentRow TYPE

packages\dtos\src\dtos\invoicePayment.dto.ts

references: PaymentType

```ts
/** Backend read model — one row from the `InvoicePayments` table. */
export type InvoicePaymentRow = {
  InvoicePaymentId: string;
  InvoiceId: string;
  BusinessId: string;
  ExternalPaymentId: string | null;
  IdempotencyKey: string;
  PaymentAmount: number;
  PaymentDate: number;
  /**
   * V-SDC PaymentType resolved at normalisation time (TAXCORE-174 Phase 1).
   * Persisted as NVARCHAR(20) — values are the string forms of the
   * `PaymentType` enum (`CASH` / `CARD` / `CHECK` / `WIRE_TRANSFER` /
   * `VOUCHER` / `MOBILE_MONEY` / `OTHER`). NULL for synthetic / pre-migration
   * rows.
   */
  PaymentType: PaymentType | null;
  /**
   * Split-tender breakdown (AMICUS multi-tender — TAXCORE-243). JSON-serialised
   * `NormalizedInvoiceTender[]` (NVARCHAR(MAX)). NULL for single-tender /
   * non-AMICUS / pre-migration rows.
   */
  TenderBreakdown: string | null;
  /** 'imported' | 'pending' | 'fiscalised' | 'error' | 'cancelled' */
  Status: string;
  FiscalInvoiceNumber: string | null;
  FiscalQrCode: string | null;
  FiscalTimestamp: number | null;
  FiscalVerificationUrl: string | null;
  FiscalSignedHash: string | null;
  FiscalRawResponse: string | null;
  /** 8-char uppercase alphanumeric sent as the V-SDC RequestId header. */
  FiscalRequestId: string | null;
  FiscalJournal: string | null;
  ErrorMessage: string | null;
  /**
   * V-SDC failure classification on `error`-state payments (migration 052,
   * TAXCORE-323 follow-up). 1 = V-SDC was unavailable (network / 5xx /
   * breaker open) → safe for the consumer's auto-rearm sweeper to re-enqueue.
   * 0 = permanent / not V-SDC (4xx rejection, cert misconfig, domain
   * validation) → never auto-retried. NULL = unknown (pre-052 rows or any
   * error path that hasn't classified yet) → treated as 0.
   */
  IsRetryable: boolean | null;
  RetryCount: number;
```

```
  EligibleForFiscalisation: boolean | null;
  FiscalisationBlockReasons: string | null;
  CancellationAttempt: number;
  CancelsPaymentId: string | null;
  /**
   * Set on the synthetic payment of a COPY invoice; FK to the source
   * payment that was copied. NULL on every other payment.
   */
  CopiesPaymentId: string | null;
  /**
   * Set on the synthetic payment of a REFUND invoice created via the
   * per-payment Refund action; FK back to the source SALE payment being
   * refunded. NULL on every other payment.
   */
  RefundsPaymentId: string | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

InvoiceRow TYPE

packages\dtos\src\dtos\invoice.dto.ts

```
export type InvoiceRow = {
  InvoiceId: string;
  BusinessId: string;
  ImportBatchId: string;
  ExternalReference: string;
  IdempotencyKey: string;
  Source: string;
  /**
   * Coarse aggregate of the invoice's payment statuses. Persisted as a
   * denormalised cache; the canonical value is `InvoiceDTO.aggregateStatus`
   * (computed from the payments array on read).
   */
  Status: string;
  InvoiceDate: number;
  DueDate: number | null;
  BuyerName: string | null;
  BuyerTin: string | null;
  BuyerAddress: string | null;
  /** Buyer email address (migration 050, TAXCORE-308). NULL on legacy / non-Sage-200
rows. */
  BuyerEmail: string | null;
  BuyerCostCenterId: string | null;
  CashierId: string | null;
  ReferentDocumentNumber: string | null;
  ReferentDocumentDT: string | null;
  CurrencyCode: string;
  SubtotalAmount: number;
  TaxAmount: number;
  TotalAmount: number;
  /** V-SDC InvoiceType string (NORMAL / PROFORMA / COPY / TRAINING / ADVANCE). Null =
default NORMAL. */
  InvoiceType: string | null;
  /** V-SDC TransactionType string (SALE / REFUND). Null = derive from totalAmount sign.
*/
  TransactionType: string | null;
  /** V-SDC PaymentType string (CASH / CARD / OTHER). Null = default CASH. */
  PaymentType: string | null;
  /** JSON-serialised Record<string, string> sent as InvoiceAdditionalFields. Null when
not set. */
  InvoiceAdditionalFields: string | null;
  /** JSON-serialised Record<string, string> sent as PaymentAdditionalFields. Null when
not set. */
  PaymentAdditionalFields: string | null;
  RawSourceData: string | null;
  /**
   * Set on a COPY invoice row; FK to the original fiscalised invoice this row
```

```
     * is a Copy Sale / Copy Refund of. NULL on regular (non-copy) invoices.
     */
    CopiesInvoiceId: string | null;
    /**
     * Set on a REFUND invoice row created via the per-payment Refund action;
     * FK to the original SALE invoice the refund is issued against. NULL on
     * everything else.
     */
    RefundsInvoiceId: string | null;
    /**
     * Epoch ms when the upstream accounting system signalled the source
     * document was voided. NULL = not voided. Stamped by `handleVoidedInvoice`
     * on the source SALE invoice; the synthetic Refund rows it then creates
     * are unaffected.
     */
    VoidedAt: number | null;
    /**
     * Xero `reference` field — surfaced as a queryable column so sync can
     * match incoming invoices back to their source quotes. Null when the
     * provider did not supply one.
     */
    Reference: string | null;
    /**
     * Set on a regular invoice that sync resolved to exactly one
     * unconverted Proforma (quote) row in the same business. FK to that
     * quote's `InvoiceId`. NULL on quotes themselves and on invoices that
     * had no matching quote.
     */
    ConvertedFromQuoteId: string | null;
    /**
     * Epoch ms when the conversion was stamped on the source quote. NULL
     * on regular invoices and on quotes that have not yet been converted.
     */
    ConvertedAt: number | null;
    /**
     * The certificate that signed this invoice (migration 042, TAXCORE-34).
     * NULL = use the business's default certificate at fiscalisation time.
     */
    CertificateId: string | null;
    /**
     * The Location this invoice was fiscalised under (migration 043,
     * TAXCORE-246). Stamped from the pushing API key's location. NULL on legacy
     * rows / OAuth sync paths — resolves to the business default downstream.
     */
    LocationId: string | null;
    /**
     * The POS till this invoice was routed through (migration 049,
     * TAXCORE-285). Stamped only on AMICUS / Sage 200 Evolution push paths
     * (resolved from the register → till routing). NULL on OAuth sync / CSV
     * paths and legacy rows. Powers the Home dashboard's POS Till filter.
     */
```

```
  TillId: string | null;
  /**
   * AMICUS POS primary key written by the on-prem agent. For a refund row
   * (TAXCORE-293), this carries the composite shape
   * `<sale>:reverse:<paymentId>` for a per-payment reversal and the bare
   * ReverseSale-id for a whole-sale void — the FE uses the `:reverse:`
   * substring to partition refunds into REVERSE PAYMENTS vs REVERSE SALES.
   * NULL on every non-AMICUS source.
   */
  PosSaleId: string | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

InvoiceSource ENUM

packages\dtos\src\enums\invoiceSource.enum.ts

```
export enum InvoiceSource {
  CsvCanonical = 'csv_canonical',
  Xero = 'xero',
  Sage = 'sage',
  Myob = 'myob',
  EposNow = 'epos_now',
  Amicus = 'amicus',
  Sage200Evolution = 'sage200evolution',
  Atrex = 'atrex',
  Http = 'http',
}
```

InvoiceStatus ENUM

packages\dtos\src\enums\invoiceStatus.enum.ts

```
export enum InvoiceStatus {
  Imported = 'imported',
  Pending = 'pending',
  Fiscalised = 'fiscalised',
  Error = 'error',
  Cancelled = 'cancelled',
  Blocked = 'blocked',
}
```

InvoiceType ENUM

packages\dtos\src\enums\invoiceType.enum.ts

```
export enum InvoiceType {
  Normal = 'NORMAL',
  Advance = 'ADVANCE',
  Training = 'TRAINING',
  Copy = 'COPY',
  Proforma = 'PROFORMA',
}
```

LocationDTO CLASS

packages\dtos\src\dtos\location.dto.ts

references: LocationDTOFields, LocationProposalStatus, LocationRow, LocationSourceProvider, LocationStatus, LocationType

```
export class LocationDTO {
  locationId: string;
  businessId: string;
  name: string;
  street: string | null;
  city: string | null;
  administrativeUnit: string | null;
  country: string | null;
  locationType: LocationType;
  status: LocationStatus;
  isDefault: boolean;
  sourceProvider: LocationSourceProvider | null;
  sourceBranchId: string | null;
  proposalStatus: LocationProposalStatus | null;
  createdAt: number;
  updatedAt: number;

  constructor(fields: LocationDTOFields) {
    this.locationId = fields.locationId;
    this.businessId = fields.businessId;
    this.name = fields.name;
    this.street = fields.street;
    this.city = fields.city;
    this.administrativeUnit = fields.administrativeUnit;
    this.country = fields.country;
    this.locationType = fields.locationType;
    this.status = fields.status;
    this.isDefault = fields.isDefault;
    this.sourceProvider = fields.sourceProvider;
    this.sourceBranchId = fields.sourceBranchId;
    this.proposalStatus = fields.proposalStatus;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: LocationRow): LocationDTO {
    return new LocationDTO({
      locationId: row.LocationId,
      businessId: row.BusinessId,
      name: row.Name,
      street: row.Street ?? null,
      city: row.City ?? null,
      administrativeUnit: row.AdministrativeUnit ?? null,
      country: row.Country ?? null,
      locationType: row.LocationType,
```

```
      status: row.Status,
      isDefault: Boolean(row.IsDefault),
      sourceProvider: row.SourceProvider ?? null,
      sourceBranchId: row.SourceBranchId ?? null,
      proposalStatus: row.ProposalStatus ?? null,
      createdAt: Number(row.CreatedAt),
      updatedAt: Number(row.UpdatedAt),
    });
  }
}
```

LocationDTOFields TYPE

packages\dtos\src\dtos\location.dto.ts

references: LocationProposalStatus, LocationSourceProvider, LocationStatus, LocationType

```
export type LocationDTOFields = {
  locationId: string;
  businessId: string;
  name: string;
  street: string | null;
  city: string | null;
  administrativeUnit: string | null;
  country: string | null;
  locationType: LocationType;
  status: LocationStatus;
  isDefault: boolean;
  sourceProvider: LocationSourceProvider | null;
  sourceBranchId: string | null;
  proposalStatus: LocationProposalStatus | null;
  createdAt: number;
  updatedAt: number;
};
```

LocationProposalStatus TYPE

packages\dtos\src\dtos\location.dto.ts

```
/**
 * Locations.ProposalStatus (TAXCORE-265).
 *  - 'proposed' — freshly upserted from a sales-agent manifest; admin
 *    hasn't reviewed yet. Surfaces in the "N locations discovered" banner.
 *  - 'rejected' — admin saw it and declined. Re-discovery never resurrects.
 *  - null — accepted / manually-created (the live state).
 */
export type LocationProposalStatus = 'proposed' | 'rejected';
```

LocationRow TYPE

packages\dtos\src\dtos\location.dto.ts

references: LocationProposalStatus, LocationSourceProvider, LocationStatus, LocationType

```
export type LocationRow = {
  LocationId: string;
  BusinessId: string;
  Name: string;
  Street: string | null;
  City: string | null;
  AdministrativeUnit: string | null;
  Country: string | null;
  LocationType: LocationType;
  Status: LocationStatus;
  IsDefault: boolean;
  SourceProvider: LocationSourceProvider | null;
  SourceBranchId: string | null;
  ProposalStatus: LocationProposalStatus | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

LocationSourceProvider TYPE

packages\dtos\src\dtos\location.dto.ts

```
/**
 * Locations.SourceProvider supported values (TAXCORE-265). Manually-created
 * locations carry `null`. Future on-premise providers (Sage 100/300/200 Evo)
 * will widen this union when their discovery flow lands.
 */
export type LocationSourceProvider = 'amicus';
```

LocationStatus TYPE

packages\dtos\src\dtos\location.dto.ts

```
export type LocationStatus = 'active' | 'inactive';
```

LocationType TYPE

packages\dtos\src\dtos\location.dto.ts

```
/**
 * Location — one row per registered point of sale (shop / branch) under a
 * business. Introduced in migration 042 (TAXCORE-34). A business owns N
 * locations; each location owns N file certificates.
 */
export type LocationType = 'Headquarters' | 'BranchOffice';
```

LoginResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

references: User

```
export type LoginResponseDTO = {
  accessToken: string;
  // refreshToken is returned in the body so native clients can store it
  // independently of the HTTP-only cookie (which is unreliable on Android).
  // Web clients use the cookie and ignore this field.
  refreshToken: string;
  businessId: string | null;
  user: User;
};
```

McpConnectionInfoDTO TYPE

packages\dtos\src\dtos\mcp.dto.ts

```
export type McpConnectionInfoDTO = {
  url: string;
  transport: 'streamable-http';
  tools: string[];
};
```

McpTokenDTO TYPE

packages\dtos\src\dtos\mcp.dto.ts

```
export type McpTokenDTO = {
  id: string;
  name: string;
  createdAt: number;
  expiresAt: number;
  lastUsedAt: number | null;
};
```

MyobFileSummary TYPE

packages\dtos\src\dtos\accounting\myobConnection.dto.ts

```
/** App-facing summary for a single MYOB company file. */
export type MyobFileSummary = {
  myobFileId: string;
  myobFileName: string;
  isActive: boolean;
};
```

MyobStartResponse TYPE

packages\dtos\src\dtos\accounting\myobConnection.dto.ts

```
/** Response shape for `GET /accounting/myob/start`. */
export type MyobStartResponse = {
  authorizationUrl: string;
};
```

MyobStatusResponse TYPE

packages\dtos\src\dtos\accounting\myobConnection.dto.ts

references: ConnectorType, MyobFileSummary

```
/** Response shape for `GET /accounting/myob/status`. */
export type MyobStatusResponse = {
  connected: boolean;
  provider: ConnectorType.Myob;
  activeFile: MyobFileSummary | null;
  /** ISO-8601 timestamp of the active connection's access-token expiry. */
  tokenExpiresAt: string | null;
  files: MyobFileSummary[];
  autoFiscaliseInvoices: boolean;
  /**
   * Epoch-ms lower bound for invoice sync. When set, all manual and scheduled
   * syncs pass this as `modifiedSince`. Null = full history (default).
   * Set via `PATCH /accounting/sync-from-date`.
   */
  syncFromDate: number | null;
};
```

NormalizedInvoice TYPE

packages\dtos\src\dtos\invoice.dto.ts

references: InvoiceSource, InvoiceType, NormalizedInvoicePayment, NormalizedLineItem, PaymentType, TransactionType

```
export type NormalizedInvoice = {
  invoiceNumber: string;
  reference?: string | null;
  source: InvoiceSource;
  invoiceDate: number;
  dueDate: number | null;
  buyerName: string | null;
  buyerTin: string | null;
  buyerAddress: string | null;
  /** Buyer email address (TAXCORE-308). Optional — null when not supplied. */
  buyerEmail?: string | null;
  /** Buyer cost-centre identifier sent to V-SDC. Optional — null when not supplied. */
  buyerCostCenterId?: string | null;
  /** Cashier / operator identifier sent to V-SDC as the `cashier` field. Optional — null
maps to default '1'. */
  cashierId?: string | null;
  /** Reference number of the original fiscalised document (corrective / refund
invoices). */
  referentDocumentNumber?: string | null;
  /** ISO 8601 date-time of the referent document (corrective / refund invoices). */
  referentDocumentDT?: string | null;
  currencyCode: string;
  lineItems: NormalizedLineItem[];
  subtotalAmount: number;
  taxAmount: number;
  totalAmount: number;
  /** Per-source-document idempotency key. SHA-256(businessId + invoiceNumber + source).
*/
  idempotencyKey: string;
  rawSourceData: Record<string, string>;
  /** V-SDC InvoiceType. Defaults to NORMAL in the mapper when not supplied. */
  invoiceType?: InvoiceType | null;
  /** V-SDC TransactionType. Defaults to SALE (or REFUND for negative totalAmount) in the
mapper when not supplied. */
  transactionType?: TransactionType | null;
  /** V-SDC PaymentType. Defaults to CASH in the mapper when not supplied. */
  paymentType?: PaymentType | null;
  /** Arbitrary key-value pairs sent as InvoiceAdditionalFields to V-SDC. */
  invoiceAdditionalFields?: Record<string, string> | null;
  /** Arbitrary key-value pairs sent as PaymentAdditionalFields inside the payment array.
*/
  paymentAdditionalFields?: Record<string, string> | null;
  /**
   * AMICUS POS primary key (sale.number / ReverseSale.Id). Persisted to
   * `Invoices.PosSaleId` by the TAXCORE-207 push extension; undefined for
```

```
   * every non-AMICUS source.
   */
  posSaleId?: string | null;
  /**
   * AMICUS register / lane id. Resolved to a till for receipt routing by the
   * TAXCORE-207 push extension. Undefined for non-AMICUS sources.
   */
  posRegisterId?: string | null;
  /**
   * Refunds only — the original sale's `posSaleId`. The TAXCORE-207 extension
   * resolves the V-SDC referent from `(BusinessId, Source='amicus', PosSaleId)`.
   */
  originalPosSaleId?: string | null;
  /**
   * In-memory only (not persisted). Carries the agent's auto-fiscalise
   * intent through `persistAndFiscaliseInvoices` so the cloud's
   * auto-dispatch gate can honour it. Default behaviour preserved when
   * absent: PROFORMA invoices are excluded from auto-dispatch; NORMAL
   * invoices dispatch as before.
   *
   * Source: AccountingInvoice.autoFiscalise → the AMICUS normaliser
   * propagates it onto NormalizedInvoice. Non-AMICUS paths leave it
   * undefined.
   */
  autoFiscalise?: boolean;
  /**
   * One entry per fiscal event. Xero invoices: one per Xero PaymentID.
   * CSV / Sage / MYOB: a single synthetic entry where
   * `paymentAmount === invoice.totalAmount` and `externalPaymentId === null`.
   * Empty array is valid — e.g. an AUTHORISED Xero invoice with no payments
   * recorded yet. Webhook UPDATEs append entries later.
   */
  payments: NormalizedInvoicePayment[];
};
```

NormalizedInvoicePayment TYPE

packages\dtos\src\dtos\invoicePayment.dto.ts

references: NormalizedInvoiceTender, PaymentType

```
export type NormalizedInvoicePayment = {
  /** Provider PaymentID (Xero PaymentID); null for synthetic CSV/Sage/MYOB rows. */
  externalPaymentId: string | null;
  /** Cash event amount in invoice currency. V-SDC fiscalises this exact value. */
  paymentAmount: number;
  /** Epoch ms of the payment event. */
  paymentDate: number;
  /**
   * SHA-256 idempotency key — populated for Xero (per-payment); empty string
   * for synthetic rows where the repo computes it post-insert from the
   * generated `InvoicePaymentId`.
   */
  idempotencyKey: string;
  /**
   * V-SDC PaymentType resolved at normalisation time from the provider's
   * payment-method data (Xero Account.BankAccountType → CARD/WIRE_TRANSFER/...
   * via `resolveXeroPaymentType` — TAXCORE-174 Phase 1). Persisted on the
   * `InvoicePayments.PaymentType` column and read by the consumer's fiscal-
   * request mapper. Null for synthetic CSV/Sage/MYOB rows (the consumer
   * falls back to `invoice.paymentType` → `CASH`).
   */
  paymentType?: PaymentType | null;
  /**
   * Split-tender breakdown (AMICUS multi-tender — TAXCORE-243). When present
   * and non-empty, the repo JSON-serialises it to `InvoicePayments.TenderBreakdown`
   * and the consumer builds the V-SDC `payment[]` wire array from it instead
   * of the scalar `paymentType`. Undefined/empty → scalar path (backward
   * compatible).
   */
  tenders?: NormalizedInvoiceTender[];
  /** Pre-flight eligibility — false blocks auto-dispatch. */
  eligibleForFiscalisation?: boolean | null;
  /** CSV of `FiscalisationBlockReason` codes when not eligible. */
  fiscalisationBlockReasons?: string | null;
};
```

NormalizedInvoiceTender TYPE

packages\dtos\src\dtos\invoicePayment.dto.ts

references: PaymentType

```
/**
 * Normaliser output — one entry per fiscal event. For Xero invoices this is
 * one entry per `Invoice.Payments[i]`; for CSV / Sage / MYOB this is a single
 * synthetic entry with `externalPaymentId = null` and
 * `paymentAmount = invoice.total` so the downstream pipeline shape is uniform
 * regardless of source.
 *
 * The `idempotencyKey` is computed by the caller (provider-specific salt — see
 *
`services/backend/src/modules/accounting/domain/mappers/accountingInvoiceToNormalized.ts`
).
 * For synthetic payments where the `InvoicePaymentId` is only known after the
 * parent row is persisted, the caller stamps the key inside
 * `IInvoicePaymentRepository.insertSynthetic`.
 */
/**
 * One resolved tender within a split-tender payment (AMICUS multi-tender —
 * TAXCORE-243). The normaliser produces these by resolving each
 * `AccountingInvoiceTender.paymentTypeCode` against the per-business
 * `paymentTypeByCode` map. Persisted as JSON on `InvoicePayments.TenderBreakdown`
 * and read by the consumer's fiscal-request mapper to build the V-SDC wire
 * `payment[]` array.
 */
export type NormalizedInvoiceTender = {
  /** Tender amount in invoice currency. */
  amount: number;
  /** Resolved V-SDC PaymentType for this tender. */
  paymentType: PaymentType;
};
```

NormalizedLineItem TYPE

packages\dtos\src\dtos\invoiceLineItem.dto.ts

```
export type NormalizedLineItem = {
  description: string;
  /** Global Trade Item Number (barcode / product code). Optional — null when not
supplied. */
  gtin?: string | null;
  quantity: number;
  unitPrice: number;
  taxRatePercent: number;
  /** NULL when no confirmed V-SDC mapping exists yet (invoice blocked for
MISSING_TAX_MAPPING). */
  taxLabel: string | null;
  lineSubtotal: number;
  lineTaxAmount: number;
  lineTotal: number;
  sortOrder: number;
  /** Arbitrary key-value pairs sent as ItemAdditionalFields to V-SDC for this line item.
*/
  itemAdditionalFields?: Record<string, string> | null;
};
```

PaymentType ENUM

packages\dtos\src\enums\paymentType.enum.ts

```
export enum PaymentType {
  Other = 'OTHER',
  Cash = 'CASH',
  Card = 'CARD',
  Check = 'CHECK',
  WireTransfer = 'WIRE_TRANSFER',
  Voucher = 'VOUCHER',
  MobileMoney = 'MOBILE_MONEY',
}
```

PrintJobDTO TYPE

packages\dtos\src\dtos\till.dto.ts

references: PrintJobStatus

```
export type PrintJobDTO = {
  id: number;
  tillId: string;
  tillName: string;
  status: PrintJobStatus;
  createdAt: string;
  printedAt: string | null;
};
```

PrintJobStatus TYPE

packages\dtos\src\dtos\till.dto.ts

```
export type PrintJobStatus = 'PENDING' | 'PRINTED' | 'FAILED' | 'STALE';
```

PublicCertificateDTO TYPE

packages\dtos\src\dtos\certificate.dto.ts

references: CertificateDTO

```
/**
 * Public shape of a CertificateDTO — strips the password/PAC plaintext so
 * the field is safe to surface in API responses. The controller wraps
 * every outgoing DTO through this helper.
 */
export type PublicCertificateDTO = Omit<
  CertificateDTO,
  'vsdcPfxPassword' | 'vsdcPac' | 'vsdcCertificateFile'
> & {
  /** Indicates whether a PFX file is on disk. */
  hasFile: boolean;
};
```

PushInvoicesResponse TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

references: ImportRowError, PushOutcome

```
/** Response body for the push endpoint. */
export type PushInvoicesResponse = {
  /** Newly-persisted invoices (idempotency-key not previously seen). */
  accepted: number;
  /** Invoices skipped as duplicates of an already-persisted row. */
  skipped: number;
  /** Per-row errors (validation / persistence failures). */
  errors: ImportRowError[];
  /**
   * AMICUS-specific per-row outcomes (empty for Sage on-prem providers). Lets
   * the agent distinguish "processed but unrouted / deferred / blocked" from a
   * transient failure it must retry.
   */
  outcomes?: PushOutcome[];
  /**
   * Sage 200 Evolution only. When `true`, the backend is requesting the agent
   * to re-read `dbo.TaxRate` and re-POST the seed payload (admin clicked
   * "Initialize tax mappings"). The agent handles this fire-and-forget and
   * clears the flag server-side via `POST /tax-mappings/seed-from-agent`.
   */
  requestTaxRateSeed?: boolean;
};
```

PushOutcome TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

references: PushOutcomeKind

```
export type PushOutcome = {
  posSaleId: string;
  outcome: PushOutcomeKind;
};
```

PushOutcomeKind TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

```
/**
 * Per-row AMICUS-specific outcome (TAXCORE-207). All three are *definitive* —
 * the invoice was processed server-side — so the agent advances its watermark
 * past the event rather than re-pushing it.
 *
 * - `register_unmapped` — persisted + fiscalised, but no print job created
 *   because the register has no mapped till.
 * - `referent_not_fiscalised` — refund deferred; auto-resolves once the
 *   original signs (`/reprocess-credit-notes`).
 * - `referent_document_missing` — refund blocked; no original found.
 */
export type PushOutcomeKind =
  | 'register_unmapped'
  | 'referent_not_fiscalised'
  | 'referent_document_missing';
```

PushProviderStatus TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

references: AgentHeartbeat

```
/** Last-push timestamp for a single on-premise provider. */
export type PushProviderStatus = {
  /** Epoch-ms ISO string of the most recent push, or null if never pushed. */
  lastPushAt: string | null;
  /**
   * Freshest agent heartbeat for this provider's API-key agent(s) (TAXCORE-321).
   * Null when no agent has reported (or the heartbeat TTL expired). The app
   * gates the backlog chip / outage badge on this being non-null.
   */
  heartbeat?: AgentHeartbeat | null;
};
```

PushStatusResponse TYPE

packages\dtos\src\dtos\api\accountingPush.api.ts

references: PushProviderStatus

```
/**
 * Response body for `GET /api/v1/businesses/:businessId/accounting/push/status`.
 * One entry per on-premise provider, derived from the most recent
 * `ImportBatch.createdAt` for that source.
 */
export type PushStatusResponse = {
  sage100: PushProviderStatus;
  sage300: PushProviderStatus;
  sage200evolution: PushProviderStatus;
  amicus: PushProviderStatus;
  /**
   * Atrex on-premise POS (TAXCORE-370). Always populated now that the backend
   * Atrex push routes + push-status wiring have shipped — the `status` builder
   * includes it in every response, same as the Sage/Amicus providers.
   */
  atrex: PushProviderStatus;
};
```

RefreshResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

references: User

```
export type RefreshResponseDTO = {
  accessToken: string;
  refreshToken: string;
  /**
   * The user's current businessId (may be null for users who have not yet
   * completed the business-onboarding step). Returned by /auth/refresh so
   * the client can restore complete authenticated state on every silent
   * refresh — without it, a web reload that triggers a silent refresh would
   * leave Redux with the persisted businessId only, which can race with
   * AuthGate evaluation and trigger a spurious onboarding redirect.
   */
  businessId: string | null;
  /**
   * The current user record. Refreshed from the DB on every refresh so role
   * changes propagate without a full re-login.
   */
  user: User;
};
```

RefundPaymentResponse TYPE

packages\dtos\src\dtos\api\invoices.api.ts

```
/**
 * Response from
 * `POST /api/v1/businesses/:businessId/invoices/:invoiceId/payments/:paymentId/refund`.
 *
 * Per-payment Refund: a new `Invoices` row is created with
 * `TransactionType='REFUND'`, `InvoiceType` inherited from the source
 * invoice (`Normal` → Normal Refund; `Advance` → Advance Refund) and
 * `RefundsInvoiceId` pointing back at the parent (source) invoice. A
 * single synthetic `InvoicePayments` row is inserted under it carrying
 * `RefundsPaymentId` pointing at the source SALE payment, and
 * `PaymentAmount` equal to the source payment's amount. The Refund
 * V-SDC submission's `referentDocumentNumber` is the source payment's
 * SDC fiscal invoice number.
 *
 * Each source payment can be refunded at most once — a second attempt
 * returns `409 INVOICE_ALREADY_REFUNDED`. The source invoice's other
 * payments can each be refunded independently (one Refund invoice per
 * refunded payment).
 *
 * `jobId` is the dispatched fiscalisation job for the new refund's
 * synthetic payment, or null when the eligibility validator blocked
 * dispatch (the refund row still exists and can be fiscalised later).
 */
export type RefundPaymentResponse = {
  /** Invoice id of the newly-created refund row (NOT the original). */
  invoiceId: string;
  /** Synthetic payment id under the new refund row. */
  invoicePaymentId: string;
  /** External reference (invoice number) of the refund row. */
  invoiceNumber: string;
  /** Fiscalisation job id, or null when dispatch was blocked at the
   *  eligibility check. */
  jobId: string | null;
};
```

RegisterResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

```
export type RegisterResponseDTO = {
  message: string;
};
```

ResetPasswordResponseDTO TYPE

packages\dtos\src\dtos\auth.dto.ts

```
export type ResetPasswordResponseDTO = {
  message: string;
};
```

SageConnectionSummary TYPE

packages\dtos\src\dtos\accounting\sageConnection.dto.ts

```
/** App-facing connection summary — never exposes tokens. */
export type SageConnectionSummary = {
  sageBusinessId: string;
  sageBusinessName: string;
  isActive: boolean;
};
```

SageStartResponse TYPE

packages\dtos\src\dtos\accounting\sageConnection.dto.ts

```
/** Response shape for `GET /accounting/sage/start`. */
export type SageStartResponse = {
  authorizationUrl: string;
};
```

SageStatusResponse TYPE

packages\dtos\src\dtos\accounting\sageConnection.dto.ts

references: ConnectorType, SageConnectionSummary

```
/** Response shape for `GET /accounting/sage/status`. */
export type SageStatusResponse = {
  connected: boolean;
  provider: ConnectorType.Sage;
  activeConnection: SageConnectionSummary | null;
  /** ISO-8601 timestamp of the active connection's access-token expiry. */
  tokenExpiresAt: string | null;
  autoFiscaliseInvoices: boolean;
  /**
   * Epoch-ms lower bound for invoice sync. When set, all manual and scheduled
   * syncs pass this as `modifiedSince`. Null = full history (default).
   * Set via `PATCH /accounting/sync-from-date`.
   */
  syncFromDate: number | null;
};
```

SetupStatusResponse TYPE

packages\dtos\src\dtos\auth.dto.ts

```
export type SetupStatusResponse = {
  isFirstUser: boolean;
};
```

TaxRateMappingDTO TYPE

packages\dtos\src\dtos\accounting\taxRateMapping.dto.ts

```ts
/** App-facing camelCase shape returned from `GET /accounting/xero/tax-mappings`. */
export type TaxRateMappingDTO = {
  id: string;
  provider: string;
  providerTaxType: string;
  /** Human-readable provider label. Null for rows confirmed before migration 015. */
  providerTaxName: string | null;
  /** Effective tax rate % at confirm-time. Null for pre-015 rows. */
  providerRate: number | null;
  vsdcTaxLabel: string;
  vsdcRate: number | null;
  /** `false` when the mapping has been soft-deleted (deactivated). */
  isActive: boolean;
  createdAt: number;
  updatedAt: number;
};
```

TillDTO TYPE

packages\dtos\src\dtos\till.dto.ts

```
export type TillDTO = {
  tillId: string;
  tillName: string;
  shopId: string;
  shopName: string;
  isOnline: boolean;
  isDefault: boolean;
  lastSeenAt: string | null;
  posRegisterId: string | null;
  /** The Location this till serves (TAXCORE-246). Null = business default. */
  locationId: string | null;
  /**
   * Canonical OS hostname this till is bound to (TAXCORE-265). For
   * AMICUS-discovered rows this is set by the sales-agent manifest before
   * any print agent runs; for legacy/manual rows it is filled at claim
   * time. Surfaced so the UI can render "Install the print agent on
   * <machineId>" copy on rows that are still awaiting a claim.
   */
  machineId: string | null;
  /**
   * True once a print agent has successfully redeemed a setup token for
   * this till (i.e. `TillTokenHash IS NOT NULL` on the row). Tills that
   * were pre-created by the AMICUS manifest but never claimed by a print
   * agent render as `false`, and the Receipt Printers UI offers an
   * "Awaiting print agent" visual instead of the normal online/offline
   * pill. The hash itself is never sent to the client.
   */
  isClaimed: boolean;
};
```

TransactionType ENUM

packages\dtos\src\enums\transactionType.enum.ts

```
export enum TransactionType {
  Sale = 'SALE',
  Refund = 'REFUND',
}
```

UnmappedTaxTypeDTO TYPE

packages\dtos\src\dtos\accounting\taxRateMapping.dto.ts

```
/** App-facing camelCase shape for unmapped tax type entries. */
export type UnmappedTaxTypeDTO = {
  id: string;
  provider: string;
  providerTaxType: string;
  /** Human-readable label from the provider. Null until a sync populates it. */
  providerTaxName: string | null;
  /** Effective tax rate %. Null until a sync populates it. */
  providerRate: number | null;
  firstSeenAt: number;
  lastSeenAt: number;
};
```

User TYPE

packages\dtos\src\dtos\auth.dto.ts

references: UserRole

```
export type User = {
  id: string;
  username: string;
  email: string;
  firstName: string;
  lastName: string;
  roles: UserRole[];
};
```

UserAdminDTO CLASS

packages\dtos\src\dtos\userAdmin.dto.ts

references: UserAdminDTOFields, UserAdminRow, UserRole

```
export class UserAdminDTO {
  userId: string;
  businessId: string | null;
  email: string;
  firstName: string;
  lastName: string;
  roles: UserRole[];
  isActive: boolean;
  lastLoginAt: number | null;
  createdAt: number;
  updatedAt: number;

  constructor(fields: UserAdminDTOFields) {
    this.userId = fields.userId;
    this.businessId = fields.businessId;
    this.email = fields.email;
    this.firstName = fields.firstName;
    this.lastName = fields.lastName;
    this.roles = fields.roles;
    this.isActive = fields.isActive;
    this.lastLoginAt = fields.lastLoginAt;
    this.createdAt = fields.createdAt;
    this.updatedAt = fields.updatedAt;
  }

  static toDto(row: UserAdminRow): UserAdminDTO {
    return new UserAdminDTO({
      userId: row.UserId,
      businessId: row.BusinessId,
      email: row.Email,
      firstName: row.FirstName,
      lastName: row.LastName,
      roles: row.Roles ? (row.Roles.split(',') as UserRole[]) : [],
      isActive: row.IsActive,
      lastLoginAt: row.LastLoginAt,
      createdAt: row.CreatedAt,
      updatedAt: row.UpdatedAt,
    });
  }
}
```

UserAdminDTOFields TYPE

packages\dtos\src\dtos\userAdmin.dto.ts

references: UserRole

```
export type UserAdminDTOFields = {
  userId: string;
  businessId: string | null;
  email: string;
  firstName: string;
  lastName: string;
  roles: UserRole[];
  isActive: boolean;
  lastLoginAt: number | null;
  createdAt: number;
  updatedAt: number;
};
```

UserAdminRow TYPE

packages\dtos\src\dtos\userAdmin.dto.ts

```
export type UserAdminRow = {
  UserId: string;
  BusinessId: string | null;
  Email: string;
  FirstName: string;
  LastName: string;
  Roles: string | null;
  IsActive: boolean;
  LastLoginAt: number | null;
  CreatedAt: number;
  UpdatedAt: number;
};
```

UserRole ENUM

packages\dtos\src\enums\userRole.enum.ts

```
export enum UserRole {
  Administrator = 'administrator',
  ViewInvoices = 'view_invoices',
  SubmitInvoices = 'submit_invoices',
  ViewAudit = 'view_audit',
  ManageUsers = 'manage_users',
  UpdateBusinessSettings = 'update_business_settings',
  UpdateTaxcoreCredentials = 'update_taxcore_credentials',
}
```

XeroStartResponse TYPE

packages\dtos\src\dtos\accounting\xeroConnection.dto.ts

```
/** Response shape for `GET /auth/xero/start`. */
export type XeroStartResponse = {
  authorizationUrl: string;
};
```

XeroStatusResponse TYPE

packages\dtos\src\dtos\accounting\xeroConnection.dto.ts

references: ConnectorType, XeroTenantSummary

```
/** Response shape for `GET /auth/xero/status`. */
export type XeroStatusResponse = {
  connected: boolean;
  provider: ConnectorType.Xero;
  activeTenant: XeroTenantSummary | null;
  /** ISO-8601 timestamp of the active tenant's access-token expiry. */
  tokenExpiresAt: string | null;
  tenants: XeroTenantSummary[];
  /**
   * Whether the business has opted in to auto-fiscalisation of invoices
   * arriving via the Xero webhook. Toggled via
   * `PATCH /api/v1/businesses/:businessId/accounting/auto-fiscalise`.
   */
  autoFiscaliseInvoices: boolean;
  /**
   * Epoch-ms lower bound for invoice sync. When set, all manual and scheduled
   * syncs pass this as `modifiedSince`. Null = full history (default).
   * Webhooks are never filtered by this value.
   * Set via `PATCH /accounting/sync-from-date`.
   */
  syncFromDate: number | null;
};
```

XeroTenantSummary TYPE

packages\dtos\src\dtos\accounting\xeroConnection.dto.ts

```
/**
 * App-facing tenant description — never exposes tokens.
 * Used by `GET /auth/xero/tenants` and `GET /auth/xero/status`.
 */
export type XeroTenantSummary = {
  xeroTenantId: string;
  tenantName: string;
  tenantType: string;
  isActive: boolean;
};
```