Client Portal + Authorize.net CIM Integration with TRUX (SQL)

Implementation Guide • Generated 2025-10-02 17:35 UTC

Executive Summary

- Goal: Deliver a modern client portal on your domain where customers can log in, view account details, pay invoices, save a payment method (via tokenization), request/schedule services, and track completions — while keeping the TRUX Haul■IT SQL database as the system of record.
- Approach: Build a small web application (ASP.NET Core + React or Razor Pages) hosted on IIS (Windows Server 2019) or AWS. Use Authorize.net Customer Profiles (CIM) so card data is never stored on your servers. Persist only profile IDs, last4, and transaction IDs. Tie all payments and service requests to TRUX via stored procedures, views, or vendor■supported import APIs.
- Security & Compliance: Prefer Authorize.net Accept Hosted for SAQ■A (lowest PCI scope). Otherwise, Accept.js (SAQ■A■EP). Enforce HTTPS, role■based access, MFA options, webhook signature validation, and least■privilege SQL accounts.

Solution Architecture (High■Level)

- Frontend: Portal UI (React or Razor Pages) with login/registration, dashboard, invoices, payments, payment methods, requests/scheduling, and history.
- Backend: ASP.NET Core Web API exposing endpoints for authentication, balances/invoices, hosted■payment token issuance, payment execution, CIM profile management, service request CRUD, and webhook receiver.
- Database: Portal DB schema (Users, Accounts, PaymentProfiles, Payments, ServiceRequests) separate from TRUX. Read TRUX via views/stored procs; write through stored procs or a supported import interface.
- Payments: Authorize.net Accept Hosted (recommended to start) or Accept.js. Use CIM to tokenize and store customer payment profiles at the gateway. No PAN/CVV at rest on your servers.
- Webhooks: Authorize.net → /api/webhooks/authorizenet for transaction lifecycle events and profile changes; idempotent processing updates Payments table and reconciles with TRUX.
- Scheduling: Portal collects request details; creates work orders/route tickets in TRUX via stored proc or import. Reads schedules and completion status via TRUX views for the dashboard.

Portal Data Model (Minimal)

Table	Key Columns	Purpose
Users	id, email, password_hash, mfa_enabled	Portal authentication and preferences.
Accounts	id, crm_account_number, display_name	Represents a TRUX customer account.
UserAccounts	user_id, account_id, role	Map users to one or more TRUX accounts.

PaymentProfiles	id, account_id, anet_customer_profile_id, anet_payment_profile_id, brand, last4, default_flag	CIM payment methods (tokenized) associated to an account
Payments	id, account_id, amount, currency, anet_transaction_id, status, created_at, settled_at, applied_invoice_ids	All portal payments and their final status
ServiceRequests	id, account_id, type, address, requested_date, status, crm_workorder_id	New requests (e.g., swap, extra pickup) synced into TRUX a

Payment Flows (Authorize.net + CIM)

1 One■Time Payment (Accept Hosted — recommended MVP):

2 1) Backend calls getHostedPaymentPageRequest to create a one■time form token (amount, invoice metadata, return URLs).

3 2) Frontend embeds the hosted form (iframe). Customer submits payment; the portal receives a response object client■side and also a webhook server■side.

4 3) Backend marks the payment as PENDING, stores Authorize.net transactionId, then finalizes to SUCCESS/FAILED upon webhook confirmation (settled/declined).

5 4) On SUCCESS, post cash receipt into TRUX and link it to invoice(s); store the TRUX receipt ID and update balance.

1 Save Payment Method (CIM):

2 1) Ensure the account has an Authorize.net customerProfileId (create if missing).

3 2) Use Accept Hosted with customerProfileId and addPaymentProfile=true (or Accept.js opaqueData) to create a paymentProfileId.

4 3) Store only profile IDs, card brand, and last4 in PaymentProfiles.

1 Charge Saved Method (One■Click Pay / Auto■Pay):

2 1) Backend initiates createTransactionRequest with profileTrans data (customerProfileId + paymentProfileId).

3 2) Persist transactionId; handle idempotency by checking if a payment with the same client token/reference already exists.

4 3) On webhook confirmation, post to TRUX and email a receipt to the customer.

TRUX (SQL) Integration

- Read (View■Only): Create SQL views or stored procs that expose: open invoices, current balance, invoice line items, next scheduled pickup(s), and recent work■order statuses for a given CRM account.
- Write (Payments): Implement a stored procedure (or vendor■provided import) to upsert a ‘cash receipt’ against invoice(s). Inputs: account number, amount, externalRef=anet_transaction_id, allocation per invoice. Return a TRUX receipt ID.
- Write (Service Requests): Expose a stored proc or import that creates a Work Order / Route Ticket (type, service address, container, notes, requested date). Return crm_workorder_id for display and tracking.

- Staging Pattern: If direct writes to TRUX tables are restricted, write requests to a staging schema (Portal.Staging_*). A scheduled SQL Agent job or Windows Service (integration worker) validates and moves data into TRUX via controlled procs, logging all changes.
- Least Privilege: Use a dedicated SQL login for the portal with EXEC rights on specific procs and SELECT on views only. No table■level write permissions to core TRUX tables.

Scheduling & Completion Visibility

- Portal UI offers: (a) upcoming pickups and open work orders; (b) a scheduler to request new services (extra pickup, swap, delivery, removal).
- When a request is submitted, create a ServiceRequests row and call the TRUX proc/import to create the actual work order (crm_workorder_id).
- Operations staff continue updating/closing work orders in TRUX (or mobile app). The portal polls or subscribes to changes (via view/proc) to reflect status: NEW → SCHEDULED → IN■PROGRESS → COMPLETED.
- Completion details (date, ticket number, photos/notes if available) are displayed on the customer’s timeline. If applicable, additional charges flow into invoices and appear on the portal once posted in TRUX.

Deployment on IIS (Windows Server 2019)

1. Install the .NET Hosting Bundle (matching your app’s runtime) and enable IIS features: WebSockets, URL Rewrite (optional), and IIS Management Console.
2. Create an IIS Site/App Pool (No Managed Code). Configure the site to reverse■proxy to Kestrel (ASP.NET Core Module handles this automatically).
3. Bind HTTPS with a valid certificate; enforce TLS 1.2+; redirect HTTP→HTTPS.
4. Set environment variables / appsettings (connection strings, Authorize.net credentials via secure secrets store).
5. Create a firewall rule for HTTPS only; restrict admin management ports to your office IP/VPN.
6. Configure Windows Event Log + rolling file logs; set up a daily backup of portal DB and config.

Webhooks, Reconciliation, and Idempotency

- Register Authorize.net webhooks (transaction.created, transaction.settled, profile.created/updated). Validate each message with the Signature Key.
- Use an IdempotencyKey for each client■initiated payment (GUID). If a duplicate request arrives, return the existing result.
- Reconciliation job: nightly job compares settled payments in Authorize.net vs. TRUX receipts; flags mismatches for review.

Security & Compliance Checklist

- Use Accept Hosted first (SAQ■A). If using Accept.js, treat site as SAQ■A■EP. Never handle or log raw PAN/CVV.
- HTTPS everywhere; HSTS; strong password policy and optional MFA; account lockouts; CSRF protection; input validation.
- SQL least■privilege: separate schema for portal tables; read■only views to TRUX; EXEC■only for integration procs; audit log all writes.
- Secrets management: avoid storing API keys in source control; use environment secrets or Windows DPAPI■protected files.
- Backups: daily DB backups + off■site copy; tested restore procedure.

Implementation Plan & Milestones (4 Weeks Example)

- Week 1: Portal skeleton (auth, user↔account link), TRUX read■only views (balance, invoices, schedule). Webhook endpoint scaffolding.
- Week 2: Accept Hosted one■time payment; store transactions; webhook■driven status; cash■receipt proc to TRUX.
- Week 3: CIM saved methods; one■click pay; auto■pay; service request forms; TRUX work order creation.
- Week 4: QA hardening (decline paths, partial allocations), reconciliation job, logging/monitoring, go■live on IIS.

Open Questions / Decisions

- Exact TRUX integration point(s): native API vs stored procedures vs import tools. Identify preferred vendor■supported path.
- Invoice■level allocation rules: FIFO, customer■selected, or auto■split across selected invoices.
- Roles/permissions: can a user manage multiple accounts/locations? Who can save payment methods?
- Service request catalog: which request types will be supported at launch (extra pickup, swap, delivery/removal, bulky item)?
- Email templates and branding for receipts and confirmations.

Appendix A — Example API Endpoints (Sketch)

- `POST /api/auth/register • POST /api/auth/login`
- `GET /api/accounts/{id}/summary • GET /api/accounts/{id}/invoices`
- `POST /api/payments/hosted/token (returns Accept Hosted token)`
- `POST /api/payments/charge (profile IDs or opaqueData)`
- `POST /api/payment-methods (create CIM payment profile)`
- `DELETE /api/payment-methods/{id}`
- `POST /api/webhooks/authorizenet (validate signature; upsert payment)`
- `GET /api/requests (list) • POST /api/requests (create)`
- `GET /api/schedule/upcoming (from TRUX views)`

Appendix B — Example SQL Permissions

- `CREATE ROLE PortalReader; GRANT SELECT ON dbo.vw_AccountSummary TO PortalReader;`
- `CREATE ROLE PortalExec; GRANT EXEC ON dbo.sp_Portal_PostCashReceipt TO PortalExec;`
- `CREATE LOGIN portal_app WITH PASSWORD = '...'; CREATE USER portal_app FOR LOGIN portal_app;`
- `EXEC sp_addrolemember 'PortalReader', 'portal_app'; EXEC sp_addrolemember 'PortalExec', 'portal_app';`