Description

Title

AI-Powered Healthcare Claims Management Platform and Methods

Background of the Invention

Healthcare providers face immense complexity in managing insurance claims due to diverse payer requirements and legacy data standards. Although the HIPAA X12 EDI formats (e.g. 837 for claims, 835 for remittances) are standardized, each payer imposes unique companion guide rules and edits. For example, one insurer may reject a claim that another would accept, due to custom field requirements or code usage . Clearinghouses have evolved to apply thousands of these payer-specific edits, scrubbing claims before submission . However, maintaining compliance with ever-changing rules is labor-intensive, and many claims are still denied for avoidable technical reasons.

Claim denials significantly impact revenue cycle performance. Studies indicate a large share of denials stem from data issues such as missing information or patient eligibility problems . Providers often must manually correct and resubmit denied claims, incurring delays and administrative costs. Traditional rules-based systems catch some errors, but they cannot easily adapt to new patterns or complex multi-factor issues. This has led to interest in AI-driven solutions that learn from historical denial patterns to flag risky claims before submission . For instance, a machine learning model can predict if a claim is likely to be denied and suggest corrective actions (e.g. adding a modifier or attaching documentation) prior to submission, reducing rework and revenue loss.

Another challenge is integrating clinical data with billing. Hospitals document services in Electronic Health Record (EHR) systems using HL7/FHIR standards, whereas billing still relies on X12 EDI transactions. Bridging these systems is cumbersome, often involving custom interfaces or manual data entry. There is a need for a platform that can ingest FHIR-based clinical data and automatically translate it into valid X12 837 claim files, and conversely handle payer responses (835/277) back to the clinical context. Moreover, certain specialized services require correlating clinical events and billing codes over time. For example, in home infusion therapy, a nursing visit billed under a temporary G-code must be linked with the drug supply billed under a J-code within a 30-day window. Existing systems often handle such code

reconciliation through manual checks or separate processes, increasing the chance of omissions or compliance errors.

Finally, attaching supporting clinical documentation and ensuring audit traceability is increasingly important. Payers may require documentation (clinical notes, operative reports, etc.) to be submitted with or linked to claims for pre- or post-payment review. Current workflows for managing attachments are largely manual (faxing or uploading PDFs) and lack a robust audit trail. Ensuring the integrity of documentation – for example by storing a cryptographic hash of each attachment in an immutable log – would provide confidence that records have not been altered . In summary, there is a clear need for an integrated claims management platform that automates payer-specific validation, leverages AI for denial prediction, reconciles related codes over time, attaches and logs documentation securely, and seamlessly connects modern clinical data exchange with legacy billing formats.

Summary of the Invention

The present invention provides a comprehensive AI-powered healthcare claims management software platform that addresses the above challenges. In one aspect, the platform automatically validates medical claims against payer-specific requirements and edits, effectively performing the same scrubbing that a clearinghouse or billing expert would. A library of configurable rules (e.g. code frequency limits, required modifiers, companion codes) is maintained and updated for each payer, enabling real-time feedback on claim compliance prior to submission . This ensures that claims meet each insurer's unique criteria, reducing initial rejections.

In another aspect, the platform includes an AI-driven denial prediction and recommendation engine. This engine analyzes both the claim's content and historical outcomes to predict the likelihood of denial for each claim or even each line item. If the risk is above a threshold, the system provides recommended corrective actions to the user – for example, suggesting the addition of a missing referral code or highlighting inconsistent data that previously led to denials. The AI model continuously learns from new data, adapting to emerging denial patterns and payer behavior. By flagging "risky" claims before they are sent out, the platform helps organizations preempt denials and improve first-pass acceptance rates . In the event a claim is denied, the same intelligence can assist in generating an appeal package (e.g. a template appeal letter citing relevant medical necessity, with all required attachments) to streamline the recovery process .

Another key feature is automated code reconciliation and charge orchestration. The invention ensures that related services and items are billed correctly across time-bound episodes. For example, for home infusion therapy, the system automatically links nursing visit codes (G-codes) with drug administration codes (J-codes) and enforces Medicare's 30-day companion rule – if a G-code is billed, the corresponding J-code must be present, and vice versa . The platform's rules engine will flag a missing companion code or even auto-generate the appropriate charge if the clinical data supports it. Similarly, the system can bundle or split charges according to payer

rules (such as per-diem rates, global periods, or episodic bundling), ensuring compliance with complex billing regulations.

The invention further provides a secure clinical documentation attachment and audit trail subsystem. Users can attach supporting documents (e.g. physician notes, consent forms, lab results) to a claim or claim line item. Each attachment is logged with a timestamp and a cryptographic hash in an immutable audit log, creating a tamper-evident record of exactly what was submitted . The system maintains provenance links from each billed charge to its source clinical data or document. For instance, every charge line can reference the specific nursing note, medication administration record, or other event that justifies it . Auditors or payer reviewers can thus access proof-of-service instantly, and any alteration to documentation would be detectable via the stored hashes. This provides end-to-end traceability and trust, reducing the administrative burden of audits.

Yet another aspect of the platform is its integration layer bridging FHIR-based inputs and X12-based outputs. The system can ingest clinical and administrative data through modern APIs (HL7 FHIR resources or SMART on FHIR integration) and use that data to generate compliant X12 837 claim files for submission . It also handles inbound EDI transactions like the 835 Electronic Remittance Advice and 276/277 claim status inquiries/responses, updating claim statuses automatically and notifying users of outcomes. By acting as a translator and orchestrator between the clinical realm and the billing realm, the invention ensures data consistency and streamlines workflows. Care providers can continue charting in their EHR, while the platform pulls the necessary data (patient demographics, encounters, orders, etc.) and produces billing claims in the required format behind the scenes .

In sum, the invention is a unified software platform comprising: (i) a validation engine with hundreds of configurable payer edits, (ii) an AI denial prediction engine that flags high-risk claims and suggests fixes, (iii) a code reconciliation module that enforces coding rules over episodes (e.g. matching temporary codes to permanent codes in required combinations), (iv) a document management and audit trail module using hashing for integrity, and (v) an integration and orchestration layer that converts and communicates data between EHR systems and payer systems (including generating X12 837/835/276 transactions). The platform is designed to be deployed either as a standalone cloud service or as an embedded add-on to existing EHR/RCM systems, thereby supporting broad use-cases from provider billing offices to third-party revenue cycle management (RCM) services. The claims are drafted to allow both narrow and broad enforcement, covering the core inventive features as well as specific implementations across different domains (e.g. hospital billing, home infusion, lab claims, patient-submitted claims, and beyond).

Brief Description of the Drawings

FIG. 1 is a high-level system architecture diagram of the AI-powered claims management platform, illustrating its integration with external systems (EHR and clearinghouse/payer systems) and core internal components.

FIG. 2A and FIG. 2B are schematic workflow diagrams of user interface interactions. FIG. 2A shows an example pre-submission workflow in which the system validates a claim in real-time and a user corrects errors and responds to AI-driven suggestions before final claim submission. FIG. 2B depicts a post-submission workflow where the system handles payer responses, including notifying the user of a denial and assisting with appeal generation.

FIG. 3 is a diagram of example data structures used in the platform, including relationships between patient records, multi-day episodes of care, clinical event records, charge line items, and an audit log.

FIG. 4 is a flow diagram illustrating the orchestration of edge functions (serverless logic modules) for tasks such as FHIR data ingestion, automated charge generation, validation, and AI scoring, coordinated via an event-driven pipeline.

FIG. 5 is a diagram of a reconciliation engine ensuring required code pairings over time-bound windows (for example, linking G-codes for services with corresponding J-codes for drugs within a 30-day interval) and producing a validated combined claim.

Detailed Description of Embodiments

System Architecture Overview

FIG. 1 illustrates an example architecture of the claims management platform and its environment. As shown, the platform (center) interfaces with upstream clinical systems and downstream payer systems. On the left, clinical data from an EHR or other source 10 is ingested via a FHIR/HL7 interface (Arrow 12). This data can include patient demographics, encounters, orders, procedure documentation, etc., which the platform normalizes and uses to create or update claim records internally. The platform's core modules (validation engine, AI engine, reconciliation logic, etc.) process the claim data as described below. On the right, the platform prepares electronic claims in X12 837 format and transmits them (Arrow 14) through a clearinghouse or gateway 20 to payer systems 30. Responses such as electronic remittance 835 or claim status 277 are received back (Arrow 16) and interpreted by the platform. In this way, the system bridges modern clinical data exchange and legacy billing networks by translating between FHIR API inputs and EDI outputs .

Within the platform, a unified database 40 stores all pertinent data – patient info, episodes of care, events, charges, rules configurations, and audit logs – allowing tight integration of clinical and financial information . Users (e.g. billing staff) interact with the system through a User Interface 50, which could be a standalone web portal or an embedded application within the EHR (such as a SMART on FHIR app). The UI 50 communicates with the platform (Arrow 18) to present dashboards, claim forms, validation feedback, and worklists. Notifications and AI-driven recommendations are pushed to users through this interface in real time. By keeping the clinicians' and billers' workflow connected (either via integration or a unified system), the invention minimizes duplicate data entry and context-switching. The architecture supports

deployment either as a cloud service interoperating with an on-premise EHR or as a module within an EHR, providing flexibility for various healthcare setups.

The core processing of the platform is modular and event-driven. For example, when new clinical data arrives (via interface 10), it triggers internal routines to update or create claims and run validations. The system uses a microservices approach, where discrete edge functions or services handle specific tasks. This is described further in FIG. 4, but at a high level the platform's orchestrator coordinates tasks such as parsing inbound data, generating charges, checking rules, and invoking the AI model. A shared data store (database 40) ensures all modules have a consistent view of the latest information, and an audit logger captures each action for traceability. The architecture thus balances integration (a single source of truth database) with decoupled functional components that can scale independently (for example, the AI engine might run on a separate compute service to handle intensive predictions). Security is maintained by keeping Protected Health Information (PHI) within the platform's controlled environment; even when advanced AI models (which might be cloud-based) are used, data can be pseudonymized or processed internally to avoid exposure .

User Interface Workflows

The platform provides an intuitive UI for revenue cycle staff to manage claims efficiently. FIG. 2A shows a representative pre-submission workflow from the user's perspective. The process begins when a new claim record is created (either automatically from incoming clinical data, or manually by a user) (Step 100). As soon as the claim is assembled, the system's validation engine runs a comprehensive rules check (Step 102) against payer-specific requirements. This includes verifying patient and insurance information, ensuring required fields (e.g. ICD-10 diagnosis codes) are present, and applying all relevant edits (for example, local coverage determination rules, coding guidelines, and the payer's companion guide rules). If any issues are found, the system flags them (Step 104), highlighting missing or incorrect data. The UI presents these flags immediately, allowing the user to correct errors or add missing information (Step 106). For instance, if a required modifier is absent for a certain CPT code per the payer's policy, the validation engine would flag it and prompt the user to add it.

After the initial rule-based validation, the AI denial prediction engine evaluates the claim (Step 108). The AI model analyzes the claim attributes in context of historical data. It might consider factors such as diagnosis/procedure mismatches, patient coverage history, unusual billing patterns, or any combination of features that in the past correlated with denials. The system then displays a risk assessment or any predicted issues (Step 110) to the user. For example, the AI might warn: "Claims of this type have a high denial rate due to missing lab results – attaching the lab report is recommended." At this stage, the user can take additional actions guided by the AI recommendations (Step 112). They might attach a document (e.g. add a physician note or test result if the AI flagged medical necessity risk), adjust a code, or include an extra note for the payer. Thanks to this two-layered approach (rules-based validation followed by AI insight), the claim is thoroughly vetted and optimized before submission .

Once the user has addressed all issues and is satisfied, they submit the claim (Step 114). The platform then converts the finalized claim data into the standard format and transmits an X12 837 file (Step 116) to the clearinghouse or payer. At this point the claim leaves the platform for payer adjudication. Notably, every user action (such as data corrections or attachments in Steps 106 and 112) and every system action (validation results, AI outputs) are logged in the audit trail with timestamps and user IDs. This ensures a complete history of how the claim was handled prior to submission, which is invaluable for compliance and later analysis.

Turning to FIG. 2B, a post-submission workflow is depicted. After the claim is sent, the platform's integration module continuously monitors for payer responses (Step 120). This can happen in real-time via the 277 Claim Status Response for inquiries or through the 835 Remittance Advice when the claim is processed . If a claim is denied or requires additional information, the platform receives the denial code and message electronically. In Step 122, the system automatically detects that a denial has occurred (for example, a CARC code in the 835 indicating denial). It immediately alerts the user of the denial and the reason (Step 124) through the UI, possibly highlighting the claim in a worklist of denied items. Simultaneously, the AI recommendation engine engages again – this time to assist with remediation. Based on the denial reason, the system may generate an appeal packet or recommended next steps (Step 126). For instance, if the denial reason is "missing clinical documentation," the system can compile the relevant records attached to the claim and draft a template appeal letter referencing them . If the denial is due to a coding issue, the system might suggest an appropriate correction (e.g. a different code or modifier) for resubmission.

The user then reviews the system's suggestions and any generated appeal documents (Step 128). They can make adjustments as needed and then submit the appeal or corrected claim through the platform. The system can package the appeal as an organized set of documentation (including the original claim data, reference number, attachments, and a cover letter). Throughout this process, the platform ensures that all actions are recorded – if the user adds a new attachment or changes a code for resubmission, these changes are hashed and logged to maintain a clear audit trail. In cases where the claim is approved and paid without issues, the right branch of FIG. 2B shows that the system will update the claim status to paid and auto-post the payment details (Step 130). Using the 835 ERA data, the platform can automatically reconcile the payment with the claim, allocate any adjustments or patient responsibility, and mark the workflow as complete (Step 132). The user sees the claim move to a paid status with all relevant details logged (Step 134). Thus, the platform covers the full lifecycle: from initial submission through either successful payment or denial and appeal, all within one interface.

Data Structures and Audit Trail

To support the above workflows, the platform employs structured data models linking clinical, financial, and audit information. FIG. 3 depicts an embodiment of these core data structures. At the top level is a Patient entity 200 which stores patient demographics (e.g. name, date of birth, insurance info). Each Patient may have one or more associated Episode records 210. An Episode represents a span of care (for example, a multi-day home infusion therapy episode, or a hospitalization case) that might encompass multiple services and claims. The Episode record

includes attributes like an episode ID, status (open/closed/on-hold), and a reference to the patient .

Within each Episode, the system records granular clinical events as EpisodeEvent entries 220. Episode events can represent actions such as a nursing visit, a medication administration, a lab result, or equipment delivery – essentially any clinically relevant occurrence during the episode. Each EpisodeEvent may contain a timestamp, a type/classification (e.g. NURSE_VISIT or REMOTE_MONITORING), notes or observations, and links to any source documents (for example, the full text of a nurse's note, or a PDF of a signed consent) . These events provide the supporting context for billing.

Charges are captured as ChargeLine records 230, each representing a billable line item on a claim. A charge line typically includes fields for the billing code (HCPCS, CPT, ICD procedure, etc.), the quantity or units, the monetary amount, and status (e.g. draft, submitted, paid, denied) . Crucially, each ChargeLine can reference an Episode (linking it to the overall case) and optionally an EpisodeEvent (linking it to the specific clinical event that justifies that charge) . For example, a charge line for infusion nursing service (HCPCS G0088) would link to the EpisodeEvent for the nurse's visit on a certain date, and a charge line for a drug administered (J-code) could link to the pharmacy dispensing event or medication administration record. By maintaining these links, the system can always trace a billed item back to clinical evidence – addressing the "proof-of-service" challenge noted earlier. If an auditor queries a charge, the billing staff can, with one click, retrieve the exact note or record that substantiates it .

Another structure shown in FIG. 3 is the AuditLog 240. This is an immutable log of all significant actions and changes in the system. Each audit log entry captures an event such as CLAIM_VALIDATED, FIELD_UPDATED, AI_RISK_SCORED, DOCUMENT_ATTACHED, etc., along with metadata: timestamp, the user or process that performed the action, and references to the entities involved (patient, episode, charge, etc.) . Importantly, for data integrity, the audit log can store cryptographic hashes of key content. For instance, when a document is attached to a charge (EpisodeEvent or ChargeLine), the system computes a SHA-256 hash of the file and stores that in the audit record. Later, to verify the document hasn't been altered, the hash can be recomputed and compared. This design provides strong integrity guarantees – any tampering with an attachment or a record would be evident by a mismatch in the audit trail . The audit log is preferably append-only: records are never modified or deleted, which is facilitated by using techniques like sequential IDs or blockchain-like ledger storage. This immutable history is invaluable for compliance (e.g. HIPAA and financial audit requirements) and dispute resolution.

Other tables not explicitly shown in FIG. 3 but present in embodiments include a Rules or configuration table for payer edits (storing the conditions for validations), a FeeSchedule for pricing (to calculate allowed amounts for codes) , and reference dictionaries (e.g. code descriptions). The platform's unified data model means all components – the validation engine, AI models, UI, etc. – draw from the same dataset. For example, the AI denial predictor can look at ChargeLine data along with linked Episode events and historical audit logs to find patterns (like frequent late attachments leading to denials). This unified design avoids data silos that plague traditional setups and makes the system more intelligent and responsive. In summary,

the data structures enable a tight coupling between clinical events, billing actions, and audit history, which is foundational to the platform's capabilities of traceability and intelligent automation.

Edge Function Orchestration and Processing Pipeline

The invention leverages an event-driven microservice architecture to perform complex claim processing tasks efficiently. FIG. 4 illustrates the orchestration of various "edge functions" or service modules in one embodiment. These functions are small, focused units of logic that can be executed, for example, in a serverless cloud environment in response to specific triggers (hence edge function). The orchestration ensures each part of the workflow (data ingestion, charge creation, validation, AI scoring, etc.) happens at the right time and in the correct sequence.

Referring to FIG. 4, when a new FHIR data bundle is received via the API (block 300), it triggers the FHIR Webhook Function 302. The FHIR webhook function parses the incoming JSON (which could contain resources like Patient, Encounter, Procedure, Observation, etc.) and transforms or maps that data into the platform's internal structures . For example, it may upsert a Patient record, create a new Episode for a hospital admission, or record EpisodeEvent entries for each clinical action noted in the FHIR bundle. After processing, this function writes the normalized data to the database and logs a receipt in the audit log (step 304). In one implementation, each incoming FHIR bundle results in an audit entry of type FHIR_RECEIVED with details such as resource count .

The completion of data ingestion then triggers the Charge Generation Function 306 (which could also be referred to as a "HCPCS engine" in the context of generating billing codes). This function wakes up when new clinical events or orders are logged (step 308). It reads the new EpisodeEvents and Episode info and applies coding rules to generate corresponding ChargeLines. For instance, if an EpisodeEvent indicates a nursing visit of a certain duration, the charge function will create the appropriate G-code line with the correct units (e.g. hours converted to units) . If a medication administration is recorded, the function generates a J-code line for the drug, pulling dosage info to determine billing units. Moreover, this function can enforce companion code rules: if a G-code is generated for a visit, it checks whether a matching J-code for drug was already present (or vice versa) and links them or raises an alert if one is missing . The output of this stage (step 310) is that the relevant charge records are created or updated in the system, still in a preliminary "draft" state associated with the Episode .

Next, the Validation/Scrubber function 312 is invoked to perform the payer-specific edit checks on the newly created or updated charges (step 314). This function consults the rules library for the payer associated with the claim and applies all relevant validations. It might run in sequence or parallel a series of checks: format validation (ensure codes are valid and active for the date of service), coverage rules (e.g., patient age vs procedure limitations), coding pairs (e.g., if procedure X then diagnosis Y required), frequency limits (e.g., code can only be billed once every N days), etc. Any violations or warnings are recorded. The function can automatically correct some issues – for example, if a trivial formatting fix is possible or a default modifier can

be added, it may do so – and it flags anything needing user attention in the claim's data (step 316). At this point, the claim data is "scrubbed" and ready for AI analysis, with annotations for any outstanding issues.

Finally, the Denial AI Function 314 is triggered (step 318). This component takes the validated claim (including any remaining rule flags) and runs the machine learning model to produce a denial risk score and recommended actions. It may call a trained predictive model which returns probabilities for different denial reasons. Based on this, the function might append an "AI note" to the claim, such as "High risk of denial for missing documentation (85% confidence)" or "Recommend verifying patient eligibility." It updates the charge or claim record with this risk metadata (step 320), which is then visible to the user as described in the UI workflow. The AI function's output and the entire sequence's results are also logged (for example, the audit log gets an entry like AI_RISK_SCORE = 0.8 for claim XYZ with recommended action).

It should be noted that the orchestration depicted ensures loose coupling: each function (302, 306, 312, 314) can be updated or scaled independently. For example, the AI model can be improved over time without altering the data ingestion logic. The use of triggers and a workflow pipeline means the system reacts to new data in near real-time: as soon as an order is entered in the EHR, the platform can ingest it, produce charges, validate them, and even predict a denial within seconds, all before the billing team even starts to review the claim. This design is highly extensible – new functions could be added for other tasks (e.g., a prior authorization check function when scheduling, or a patient cost estimation function) and plugged into the event stream. The orchestration can be implemented using cloud function queues, a rules engine, or a directed acyclic graph (DAG) workflow tool. The inventive aspect is the specific combination of these functions tailored to claims management tasks and the event-driven coordination that mirrors the actual lifecycle of claim data.

Reconciliation Engine for Code Matching

One specialized embodiment of the validation subsystem is the reconciliation engine for paired codes, illustrated in FIG. 5. This component ensures that certain billing codes which are clinically dependent on each other are properly captured within the allowed time window and across claim formats. As an example scenario, consider the home-infusion therapy context: Medicare requires that a home nursing visit (billed under HCPCS G0068/G0088 series codes) is linked to an infusion drug administration (billed under J-codes) during a 30-day period, otherwise the service may not be reimbursable . The platform's reconciliation engine automates this cross-check.

Referring to FIG. 5, the engine takes as input the set of service codes (e.g. G-codes) 400 detected for a given patient/episode and the set of item/drug codes (e.g. J-codes) 402 for the same timeframe. Using the rules configured (which can specify, for instance, that code G0089 "Home infusion, subsequent visit" requires at least one J code in the preceding 30 days for a drug, and that J-code drugs should have a corresponding G-code visit within the same episode), the engine performs a matching (block 410). It will pair up codes that satisfy the criteria and identify any unmatched codes. For example, if a J-code drug was billed but no nursing visit was

recorded in that episode, this would be flagged as a potential compliance issue. Conversely, if a nursing visit G-code is present without any drug charge, the engine might infer that a drug was administered but not recorded, and prompt for adding the appropriate J-code. In some cases, the engine can even auto-generate the missing complementary charge: the platform might automatically insert a placeholder J-code charge if clinical data confirms a drug was given, linking it to the nursing visit event .

The output of the reconciliation engine is a set of validated claim data with all required code pairs present 412. If everything checks out (each G-code has a matching J-code within the window, and vice versa), the claim is marked as consistent. If not, the claim remains in a draft or hold state and the UI will alert the user to the discrepancy. This feature drastically reduces manual auditing of accounts to catch these issues. It is especially useful in scenarios like infusion, durable medical equipment (where certain supply codes require service codes), or surgery (where certain procedures require an assistant surgeon code or an anesthesia code to match). The engine can be generalized to any rule of the form “Code A must co-occur with Code B within X days”. Those rules are part of the payer edit library and can be configured per payer or universally for government payers like Medicare. The invention thus proactively enforces coding compliance that spans multiple claim lines or even multiple claims.

Furthermore, this reconciliation logic can handle time-bound bundling. For instance, if a payer has an episodic payment model (say, all services within 30 days of a surgery are bundled into one payment), the engine can ensure that those services are grouped appropriately and not billed separately. It would detect that certain codes within the time window should not be billed in addition to a global surgical fee, thus either preventing those charges or tagging them as non-billable to avoid denial. These are the kinds of nuanced rules that the platform encapsulates and automates. By doing so, it drastically lowers denial rates due to technical billing issues, as evidenced by initial deployments which saw significantly fewer rejections once G-code/J-code mismatches and similar problems were eliminated.

Additional Embodiments and Use Cases

While the above description often references a home-infusion therapy implementation (since it exemplifies many complex requirements), the invention is not limited to that domain. The platform’s design is broadly applicable across healthcare billing scenarios. For example, in laboratory billing, the system could attach lab result documents to claims for high-cost genetic tests and hash them in the audit log to prove result authenticity. The AI predictor might flag tests likely to be denied for lacking prior authorization, prompting staff to verify approvals before submission. In real-time eligibility checking, an embodiment might integrate a 270/271 eligibility verification step into the workflow, automatically querying insurance coverage when patient data is received and alerting if coverage is inactive – preventing a common cause of denial . In patient-submitted claims (e.g. out-of-network reimbursement requests), the platform could be used by patients or third-party services to validate that all necessary information (like referral letters or receipts) are included, and to predict any issues before the patient mails the claim to their insurer. The claims and systems described herein are intended to cover such variations. The modular nature of the platform means features can be enabled or disabled per use-case –

for instance, a hospital might use the full spectrum (validation, AI, attachments, etc.), whereas an RCM outsourcing service could use just the AI prediction and worklist on top of their existing billing software, by feeding claim data into the AI engine via API.

The claims below are drafted to capture the inventive subject matter in various forms – as methods, systems, and computer-readable media – and include both broad independent claims and narrower dependent claims covering specific embodiments. It will be appreciated by one skilled in the art that various changes can be made without departing from the scope of the invention, which is defined by the claims.

SUPPLEMENT TO THE PCT SPECIFICATION

"INTEGRATED DRUG-WASTE, VALIDATION & CLEARING-HOUSE SUBMISSION PLATFORM"

(Continuation-in-part to the parent application you drafted earlier)

I. FIELD OF THE INVENTION

The present disclosure relates to computerized health-care revenue-cycle systems, and more particularly to architectures that (i) capture and reconcile drug-waste events in real-time, (ii) auto-generate CMS-mandated JW/JZ modifiers, (iii) gate claim readiness through multi-factor validation, and (iv) submit fully compliant 837 batch files-together with 276/277 feedback loops-to one or more clearing-house exchange endpoints.

II. CROSS-REFERENCE TO FIGURES (ADDITIONAL)

- **FIG. 6** - High-level component diagram showing Waste-Capture Pipeline (pharmacy **610** and home-infusion **620**) feeding Validation Engine **630** and Charge Composer **640**.
- **FIG. 7** - Sequence diagram of compounding event (SDV vial) producing administered dose and auto-calculated waste line (JW) **710** or zero-waste line (JZ) **712**.
- **FIG. 8** - UI wireframe of Provider / Payer Administration panels (**800-A**, **800-B**) with credential store **805**.
- **FIG. 9** - CRON-scheduled 837 batch exporter **900** posting via SFTP/API to clearing house **910**, followed by 835 ingest **920** and status reconciliation loop **930**.

- **FIG. 10** - Database schema extension (normalized) for provider_profiles, payer_profiles, fee_schedules, waste_events, and claim_validations.

(Numbers continue after FIG. 5 of the parent spec.)

III. SUPPLEMENTAL SUMMARY

A. Waste-Aware Charge Composer

1. **Dual-Context Capture** –
   Pharmacy Context: When a compound is logged, the preparer records "volume prepared" directly on the CompoundingPanel.
   Administration Context: At home or bedside, the nurse records "volume administered" and, if infusion ends prematurely, any "volume returned."
2. **Auto-Derived Waste Logic** –
   System computes
   WASTE = Prepared – Administered
   Then determines:
   - **If WASTE > 0 & SDV → add JW line**
   - **If WASTE = 0 & SDV → add JZ line**
   - **If vial is MDV → no JW/JZ, flag for zero-waste compliance audit**
3. **Hazardous Waste Flag** – Each waste event inherits is_hazardous from the NIOSH / USP <800> classification in inventory and attaches disposal-log metadata for regulatory recall.
4. **Linkage** – The waste event foreign-keys to both the Episode **(fx_episode_id)** and the concrete ChargeLine rows.

B. Provider / Payer Administration

- UI panels (FIG. 8) allow super-users to persist:
  - **Rendering provider** taxonomy, NPI, TIN, facility NPI, POS codes.
  - **Billing provider** override (if different from rendering).
  - **Payer** objects holding payer name, EDI ID, payer-assigned code sets, claim-specific rules and required attachments.

C. Fee-Schedule Engine

- Table fee_schedules maps (payer_id, hcpcs_code, modifier, eff_date) → *allowed amount*.
- Hook useFeeSchedule() caches look-ups so that Charge Composer writes the adjudication-ready dollar value into charge_amount before 837 serialization.

D. Multi-Factor Validation Engine

- New microservice (Edge Function claimmd-clearinghouse/validate-claim.ts) walks the Episode graph and asserts:

1. **Provider Completeness** (NPI, TIN, taxonomy).
2. **Payer Specific** attachment rules (e.g., IVIG supporting labs).
3. **JW/JZ Presence** for every SDV drug line.
4. **Fee-Schedule Hit** for each HCPCS+modifier tuple.

- Failing rules populate claim_validation_errors and block CRON export.

E. CRON-Driven Batch Export & 276/277 Polling

- pg_cron job fires nightly 02:00 local:
    1. Select Episodes whose validation status is "PASS" and not yet exported.
    2. Edge Function claimmd-clearinghouse/export-837.ts builds X12 5010 837P/I/D file with full loop population (NM1, REF, SBR, CLM, SV1, LIN, DTP, etc.).
    3. Output transmitted via **SFTP or REST** (configurable per payer) using stored credentials.
    4. Job inserts tracking row in claim_exports.
- Separate cron every 6 h hits Claim.MD 276 API (or other) per export to pull 277 status; maps ACK/ERRs into claim_status_updates and triggers UI badges.

IV. DETAILED EMBODIMENT (WITH CODE POINTERS)

1. **Waste RPC (fn_get_dose_and_waste)** - PostgreSQL function enumerated in FIG. 10 table **waste_events**, already unit-tested with 100 mg/90 mg example.
2. **Edge Function** waste-charge-sync/index.ts (supabase/functions) implements FIG. 7 logic; discussed in commit *d1f7be3*.
3. **React Hook** useWasteSync() (src/hooks/useWaste.ts) provides front-end mutation, automatically passing Auth JWT.
4. **Validation Engine** - claimmd-clearinghouse/validate-claim.ts uses rule set JSON fetched from payer_profiles.validation_rules.
5. **837 Exporter** - claimmd-clearinghouse/export-837.ts re-uses the open-source *libx12* serializer; loops dynamically constructed from Episode→ChargeLine graph.

CRON Schedules - Registered via SQL:

```
-- Nightly export @ 02:00

SELECT cron.schedule(

  'nightly_837_export',

  '0 2 * * *',

  $$ call public.run_837_export() $$

);

-- Six-hourly 276 polling
```

```
SELECT cron.schedule(
  'poll_276_277',
  '0 */6 * * *',
  $$ call public.poll_claim_status() $$
);
```

6.
7. **UI Enhancements -**
    - **Waste dialogs** auto-calculate waste and preview JW/JZ before commit.
    - **Validation Banner** atop Episode detail shows PASS / FAIL with drill-down.
    - **Provider/Payer Admin panels** (FIG. 8) under /settings.

V. ADVANTAGES OVER PRIOR ART

- **Zero-click modifier accuracy** - human enters only prepared/administered units; software guarantees correct JW/JZ coding.
- **Unified clearing-house pipeline** - same data objects flow from clinical capture → validation → X12 serialization → 276/277 reconciliation, eliminating dual-entry errors.
- **Dynamic fee-schedule** resolution permits payer-specific payment prediction **prior** to submission.
- **Hazardous-waste compliance** links clinical, financial, and environmental documentation in one ledger record.

VI. CLAIMS ROADMAP (for future continuation)

Method, apparatus, and non-transitory computer-readable medium claims directed to:

1. Auto-derivation of drug waste and modifier insertion.
2. Rule-based validation gating using payer-specific JSON schema.
3. Dynamic, schedule-driven X12 batch generation with loop templates populated from graph database traversal.
4. Bidirectional state machine integrating 276/277 acknowledgments to update Episode financial state.

(Exhaustive claim language will be appended in the next formal draft.)

VII. CONCLUSION

The supplemental architecture embeds compounding- and administration-aware waste capture, automated modifier coding, provider/payer master data, validation gating, and fully automated 837/276/277 clearing-house integrations-all without additional user burden-thereby closing the gap between clinical workflow and compliant reimbursement.