Founding Engineer Challenge

Oct 10 - Oct 16, 2025

Context

At RPG, we build accounting automation tools that help accountants work faster and more accurately. In this challenge, you're building a core component of accounting workflows: document understanding. We're evaluating your engineering judgment, code quality, and ability to navigate ambiguity.

Objective

Build a simple workflow that goes from Client → Intake → Document → Classify → Extract → Checklist.

- Client = an individual whose complexity determines expected document count.
- Intake = a tax-year case (e.g., fiscal_year = 2025) representing one work session for that client.

Show us how you design clean systems, make trade-offs, and ship maintainable code fast.

Scope (what to build)

Create an independent FastAPI service that can:

1. Create Client
    - Accept client information and complexity level
    - Complexity determines expected document volume:
        - simple (T4 + id); average (T4 + id + 2 receipts); complex (T4 + id + 5 receipts)
    - Complexity is frozen at client creation
2. Create Intake
    - Accept client_id and fiscal_year
    - Initialize checklist based on the client's complexity
    - Start with status = open
3. Ingest Documents
    - Upload documents via API to a specific intake
    - Accept: PDF, PNG, JPG
    - Store files locally in a /bucket directory
    - Calculate and store SHA256 hash for each file
    - Reject duplicate uploads within the same intake (same hash)
    - Return Document ID on successful upload
4. Classify Documents
    - Detect document kind for all unknown documents in an intake
    - Document kinds: T4, receipt, id, or unknown

    - This may be a long-running operation
    - Update checklist when documents are classified
5. Extract Data
    - Parse 2-3 key fields from each classified document (T4, id, receipt)
    - Example fields to extract:
        - T4: employer_name, box_14_employment_income, box_22_income_tax_deducted
        - id: full_name, date_of_birth, id_number
        - receipt: merchant_name, total_amount
    - Store extracted data as JSON in the Document record
    - Mark corresponding checklist items as received after successful extraction.
    - If all items are received, set intake status = done
6. Checklist Management
    - Track which required documents are missing vs. received
    - Automatically update intake status when all required items are received

Minimal Domain Models

- Client
    - id (int/uuid)
    - name (string)
    - email (string)
    - complexity (enum: simple | average | complex)
    - created_at (timestamp)
- Intake
    - id (int/uuid)
    - client_id (foreign key)
    - fiscal_year (int, e.g., 2025)
    - status (enum: open | done)
    - created_at (timestamp)
- Document
    - id (int/uuid)
    - intake_id (foreign key)
    - filename (string)
    - sha256 (string, 64 chars)
    - mime_type (string)
    - size_bytes (int)

    - stored_path (string)
    - uploaded_at (timestamp)
    - doc_kind (enum: T4 | receipt | id | unknown, defaults to unknown)
- ChecklistItem
    - id (int/uuid)
    - intake_id (foreign key)
    - doc_kind (enum: T4 | receipt | id) - the expected document type
    - status (enum: missing | received)

API Endpoints

- POST /clients → create a client
- POST /intakes → create intake and initialize checklist
- POST /intakes/{id}/documents → upload document to intake
- POST /documents/{document_id}/classify → classify a single document
- POST /intakes/{id}/classify → classify all unknown documents in the intake
- POST /documents/{document_id}/extract → extract for a single document
- POST /intakes/{id}/extract → extract key fields from all classified documents in the intake
- GET /intakes/{id}/checklist → get current checklist status - returns: Checklist items + intake status

Stack: Python 3.11+, FastAPI, Uvicorn, Pydantic. Choose an appropriate database. For the classification task, we encourage exploring open-source models, but do not use paid APIs.

Sample Documents Provided

We're including sample documents in /sample_docs:

- T4_sample.pdf - Canadian T4 tax form (should classify as T4)
- T4_sample.jpg - Canadian T4 tax form (should classify as T4)
- drivers_license.jpg - ID document (should classify as id)
- cat.jpg - Random image (should remain unknown)
- /receipts - Scanned and digital receipts (should classify as receipt)

Use these for testing your classification logic.

Deliverables

- Public GitHub repository
- Clear README with a short overview, architecture explanation, and instructions to run the service
- At minimum, two tests covering:
    1. upload → classify → extract → checklist flow
    2. Duplicate upload detection using the same file hash
- What's next: what would you build with more time?

Evaluation Matrix

Category	**What We're Looking For**	**Weight**
System Design & Architecture	Clean domain modeling, sensible database choice, modular structure, and clear separation of concerns.	25%
Code Quality & Maintainability	Readable and well-organized. Thoughtful naming, error handling. Well-commented.	25%
Workflow Implementation	Correct and complete flow: client → intake → upload → classify → extract → checklist updates	25%
Creativity & Technical Judgment	Smart trade-offs, reasonable simplifications, and interesting design or ML decisions for classification/extraction.	15%
Testing & Documentation	Tests that demonstrate functionality, plus a clear README explaining architecture and how to run the app.	10%

Good luck!

– The RPG Team