CollectIQ — Backend Project Specification

Purpose: Define the backend architecture and implementation using AWS Lambda behind Amazon API Gateway, provisioned via Terraform. This doc covers project setup, coding standards, auth, APIs, data model, real-time pricing ingestion, authenticity pipeline, multi-agent orchestration, testing, observability, and interfaces with DevOps.

# 1. Architecture Overview • Entry: Amazon API Gateway (HTTP API) terminates requests and routes to Lambda functions. • Auth: API Gateway JWT Authorizer (Cognito User Pool) validates tokens before invoking Lambdas. • Compute: Node.js 20 Lambdas (TypeScript) for presign, cards CRUD, revalue (orchestration entry), pricing adapters, authenticity. • Orchestration: AWS Step Functions coordinates tasks; it invokes Amazon Rekognition first for CV/OCR feature extraction, then calls Amazon Bedrock (Supervisor/Sub‑agents) for reasoning. Events flow via EventBridge. • Storage: S3 (uploads, private), DynamoDB (single-table), Secrets in SSM/Secrets Manager. • AI: Rekognition runs BEFORE Bedrock to produce a FeatureEnvelope (OCR + visual signals). Bedrock consumes that envelope to generate authenticity scores and valuation reasoning.

# 2. Tech Stack & Principles

• Language/Runtime: TypeScript on Node.js 20 (ESM), not strict.

• Build: esbuild; one handler per Lambda; optional shared layer for utils.

• Validation: Zod at the edge with type inference inward.

• Error Model: RFC 7807 (problem+json).

• Security: Cognito JWT authorizer; least-privilege IAM; presigned S3; KMS at rest; HTTPS-only.

• Performance: P95 < 400ms hot paths; long tasks via Step Functions.

• Reliability: idempotency keys on POST; retries/backoff; circuit breakers for sources.

# 3. Repository & Code Layout

packages/backend/  
 src/  
 handlers/ (apiGateway proxy)  
 upload\_presign.ts  
 cards\_create.ts  
 cards\_list.ts  
 cards\_get.ts  
 cards\_delete.ts  
 cards\_revalue.ts  
 adapters/ (eBay, TCGPlayer, PriceCharting, Rekognition, Bedrock)  
 auth/  
 pricing/  
 authenticity/  
 store/  
 utils/  
 tests/  
infra/terraform/ owned by DevOps; Backend supplies IAM needs & env var contracts.

# 4. Environment & Configuration

# 5. Authentication & Authorization

• API Gateway JWT authorizer (Cognito) required for all routes except /healthz.

• Handlers re-parse claims to extract sub/email; enforce ownership on every read/write.

• 401/403 return RFC 7807 with remediation text.

# 6. Data Model (DynamoDB Single-Table)

Table: {stage}-CollectIQ  
PK = USER#{sub}  
SK = CARD#{cardId} (card) or PRICE#{iso8601} (snapshot)  
Attrs: userId(sub), cardId, name, set, number, rarity, conditionEstimate, frontS3Key, backS3Key, idConfidence, authenticityScore, authenticitySignals{}, valueLow, valueMedian, valueHigh, compsCount, windowDays, sources[], createdAt, updatedAt  
GSIs: GSI1(userId#createdAt), GSI2(set#rarity).

# 7. APIs (Lambda + API Gateway)

• POST /upload/presign → upload\_presign.ts  
• POST /cards → cards\_create.ts  
• GET /cards → cards\_list.ts  
• GET /cards/{id} → cards\_get.ts  
• DELETE /cards/{id} → cards\_delete.ts  
• POST /cards/{id}/revalue → cards\_revalue.ts (starts Step Functions; 202)  
• GET /healthz → public

# 8. Real-Time Pricing Ingestion & Fusion

interface PriceSource { fetchComps(q: PriceQuery): Promise<RawComp[]> }  
Normalize (conditions, currency, IQR), Fuse (low/median/high, compsCount, volatility, confidence), Cache (in-memory + DDB snapshot), rate-limit/backoff, circuit breaker.

# 9. Authenticity Pipeline Rekognition is invoked first to extract visual/ocr features and produce a FeatureEnvelope used by Bedrock: FeatureEnvelope := { ocr: [...], borders: {ratios, symmetry}, holoVariance: number, fontMetrics: {kerning, alignment}, quality: {blur, glare}, imageMeta: {...} } Bedrock then combines these signals with card metadata to produce authenticityScore ∈ [0,1] with a short rationale. Scores < 0.85 are flagged; component signals are persisted for transparency and future learning.

# 10. Orchestration (Multi-Agent) cards\_revalue starts a Step Functions execution with input { userId, cardId, s3Keys{front,back?}, requestId }. Workflow: • Task 1 — RekognitionExtract: Lambda calls Rekognition and returns FeatureEnvelope. • Parallel — PricingAgent and AuthenticityAgent (Bedrock) run concurrently; AuthenticityAgent consumes the FeatureEnvelope. • Join — Aggregator merges results, persists to DynamoDB, emits EventBridge events. • Error handling — retries/backoff per task, DLQ for persistent failures, partial result persistence allowed.

# 11. Coding Standards & Patterns

• Thin handlers → domain → adapters; pure functions; ProblemDetails; idempotency via DDB tokens; structured JSON logs (no PII).

# 12. Observability

CloudWatch logs/metrics; X-Ray traces; dashboards per endpoint/agent; alarms via DevOps.

# 13. Testing

Unit, integration (S3/DDB/Step Functions with mocks), E2E API (auth→upload→identify→value→save), error/rate-limit scenarios.

# 14. Interfaces with DevOps

Provide handler names, IAM needs, env var names, memory/timeout hints, optional OpenAPI; consume Terraform outputs.

AWS\_REGION=  
DDB\_TABLE=  
BUCKET\_UPLOADS=  
COGNITO\_USER\_POOL\_ID=  
COGNITO\_CLIENT\_ID=  
JWKS\_URL=  
ALLOWED\_UPLOAD\_MIME=image/jpeg,image/png,image/heic  
MAX\_UPLOAD\_MB=12  
EBAY\_APP\_ID= (SSM/Secrets)  
TCGPLAYER\_PUBLIC\_KEY= (SSM/Secrets)  
TCGPLAYER\_PRIVATE\_KEY= (SSM/Secrets)  
PRICECHARTING\_KEY= (SSM/Secrets)  
CACHE\_TTL\_SECONDS=300  
IDEMPOTENCY\_TTL\_SECONDS=600  
STEP\_FUNCTIONS\_ARN=  
EVENT\_BUS\_NAME=  
BEDROCK\_MODEL\_ID=<e.g., anthropic.claude-3-sonnet>  
BEDROCK\_MAX\_TOKENS=2048  
BEDROCK\_TEMPERATURE=0.2