CollectIQ — Engineering Requirements Document

AI-powered trading card intelligence — real-time valuation, authenticity, and collector trust.

**Hackathon Edition — Engineering Build Spec**Version 1.0  
Team: CollectIQ Project Team  
Date: October 14, 2025

# Table of Contents

1. Purpose & Scope
2. System Overview
3. Non‑Functional Requirements (SLOs)
4. Detailed Architecture & Data Flows
5. Interfaces & API Contracts
6. Data Model (DynamoDB)
7. Orchestration (Step Functions)
8. Security, IAM & Compliance
9. Observability & SRE
10. Environments, CI/CD & Deployment
11. Test Strategy & Acceptance
12. Risks & Mitigations
13. Traceability (PRD → ERD)
14. Appendix A: Sample Payloads
15. Appendix B: Example Policies & Terraform Stubs

# 1. Purpose & Scope

This ERD specifies the concrete engineering requirements for CollectIQ. Optimized for fast delivery and demo reliability under hackathon time constraints.

Scope includes API definitions, data flows, AWS service responsibilities, performance targets, security constraints, and test criteria.

# 2. System Overview

Architecture: API Gateway (JWT authorizer) → Lambda handlers → Step Functions. Rekognition runs FIRST to extract visual signals (FeatureEnvelope), then Bedrock performs authenticity/valuation reasoning. Pricing adapters fetch comps in parallel; Aggregator persists to DynamoDB and emits EventBridge events. Frontend is Next.js on Amplify.

| Service | Responsibility | Primary Owner |
| --- | --- | --- |
| API Gateway (HTTP API) | Public entrypoint, JWT auth, routing | DevOps |
| Lambda (Handlers) | Presign, CRUD, Revalue, Agents | Backend |
| Step Functions | Multi‑agent workflow orchestration | DevOps |
| Rekognition | OCR + visual features | Backend |
| Bedrock | Reasoning (authenticity/valuation) | Backend |
| DynamoDB | Cards, valuations, signals storage | Backend |
| S3 | Image uploads (private) | Backend |
| EventBridge | Domain events | DevOps |
| Cognito | User auth | DevOps |
| Amplify Hosting | Frontend hosting | DevOps |
| CloudWatch/X‑Ray | Logs, metrics, tracing | DevOps |

# 3. Non‑Functional Requirements (SLOs)

| SLO | Target |
| --- | --- |
| API latency (p95) | ≤ 400 ms for CRUD; ≤ 2.8 s for revalue (end‑to‑end) |
| Availability | ≥ 99.5% hackathon / ≥ 99.9% venture |
| Throughput | ≥ 10 RPS hackathon / ≥ 100 RPS venture (burst) |
| Cost ceiling | ≤ $300/mo hackathon / budgeted per growth stage venture |
| Security | JWT auth, TLS‑only, least‑privilege IAM, KMS at rest |
| Privacy | No PII beyond Cognito claims; redact tokens in logs |

# 4. Detailed Architecture & Data Flows

Data Flow (Revalue):

1) Frontend → POST /cards/{id}/revalue (Authorization: Bearer <JWT>)  
2) API Gateway → Lambda (cards\_revalue) validates & starts Step Functions execution  
3) Step Functions:  
 a. Task RekognitionExtract → returns FeatureEnvelope  
 b. Parallel:  
 - PricingAgent → fetch & normalize comps  
 - AuthenticityAgent → invoke Bedrock with FeatureEnvelope  
 c. Aggregator → persist to DDB, emit EventBridge  
4) GET /cards/{id} returns latest valuation & authenticityScore

# 5. Interfaces & API Contracts

All requests require Cognito JWT except /healthz. Errors use RFC7807.

| Endpoint | Method | Auth | p95 Target | Notes |
| --- | --- | --- | --- | --- |
| /upload/presign | POST | JWT | ≤ 200 ms | Returns {url,key}; key=uploads/{sub}/{uuid}.{ext} |
| /cards | POST | JWT | ≤ 300 ms | Create/update card; Idempotency‑Key header |
| /cards | GET | JWT | ≤ 300 ms | List (paginated) |
| /cards/{id} | GET | JWT | ≤ 300 ms | Ownership enforced |
| /cards/{id} | DELETE | JWT | ≤ 300 ms | Ownership enforced |
| /cards/{id}/revalue | POST | JWT | ≤ 2.8 s | Starts Step Functions; returns 202 |
| /healthz | GET | None | ≤ 100 ms | Liveness |

Representative Schemas:

Request: POST /upload/presign

{  
 "contentType": "image/jpeg",  
 "fileExt": "jpg"  
}

Response:

{  
 "url": "https://s3.amazonaws.com/....",  
 "key": "uploads/USER#<sub>/<uuid>.jpg"  
}

Request: POST /cards/{id}/revalue

{  
 "s3Keys": { "front": "uploads/USER#<sub>/<uuid>.jpg", "back": "uploads/USER#<sub>/<uuid2>.jpg" },  
 "windowDays": 30  
}

ProblemDetails (error):

{  
 "type": "https://docs.collectiq.app/errors/validation",  
 "title": "Validation failed",  
 "status": 400,  
 "detail": "contentType must be one of [image/jpeg,image/png,image/heic]",  
 "instance": "/upload/presign"  
}

# 6. Data Model (DynamoDB Single‑Table)

| PK | SK | Entity | Attributes (partial) |
| --- | --- | --- | --- |
| USER#{sub} | CARD#{cardId} | Card | name,set,number,rarity,conditionEstimate,frontS3Key,backS3Key,idConfidence,createdAt,updatedAt |
| USER#{sub} | PRICE#{iso8601} | Valuation Snapshot | valueLow,valueMedian,valueHigh,compsCount,windowDays,confidence,authenticityScore,authenticitySignals,sources[] |

Indexes: GSI1(userId#createdAt) for listing; GSI2(set#rarity) for analytics.

# 7. Orchestration (Step Functions ASL excerpt)

{  
 "Comment": "Rekognition → Parallel(Pricing, Authenticity) → Aggregator",  
 "StartAt": "RekognitionExtract",  
 "States": {  
 "RekognitionExtract": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke",  
 "Parameters": { "FunctionName": "rekognition\_extract", "Payload.$": "$" },  
 "ResultPath": "$.features",  
 "Next": "ParallelAgents",  
 "Retry": [{ "ErrorEquals": ["States.ALL"], "IntervalSeconds": 2, "MaxAttempts": 3, "BackoffRate": 2 }]  
 },  
 "ParallelAgents": {  
 "Type": "Parallel",  
 "Branches": [  
 { "StartAt": "PricingAgent",  
 "States": { "PricingAgent": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke",  
 "Parameters": { "FunctionName": "pricing\_agent", "Payload.$": "$" },  
 "End": true }}},  
 { "StartAt": "AuthenticityAgent",  
 "States": { "AuthenticityAgent": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke",  
 "Parameters": { "FunctionName": "authenticity\_agent", "Payload.$": "$" },  
 "End": true }}}  
 ],  
 "Next": "Aggregator"  
 },  
 "Aggregator": {  
 "Type": "Task",  
 "Resource": "arn:aws:states:::lambda:invoke",  
 "Parameters": { "FunctionName": "aggregator", "Payload.$": "$" },  
 "End": true  
 }  
 }  
}

# 8. Security, IAM & Compliance

• Cognito JWT required for all routes except /healthz. • API Gateway JWT authorizer enforces iss/aud/exp/token\_use. • S3 presign whitelist MIME & size. • KMS at rest; TLS in transit. • Least-privilege IAM per function.

Example IAM policy (RekognitionExtract Lambda):

{  
 "Version": "2012-10-17",  
 "Statement": [  
 { "Effect": "Allow", "Action": ["rekognition:DetectText","rekognition:DetectLabels"], "Resource": "\*" },  
 { "Effect": "Allow", "Action": ["s3:GetObject"], "Resource": "arn:aws:s3:::collectiq-uploads/\*" },  
 { "Effect": "Allow", "Action": ["dynamodb:PutItem","dynamodb:UpdateItem"], "Resource": "arn:aws:dynamodb:\*:\*:table/collectiq-\*" }  
 ]  
}

# 9. Observability & SRE

Structured JSON logs; X‑Ray traces across API→Lambda→Step Functions; Alarms: API 5xx, Lambda errors/duration, Step Functions failed, DLQ depth; Dashboards for p50/p95, throttles, costs.

# 10. Environments, CI/CD & Deployment

Terraform for infra; Amplify for frontend. Backend pipeline: lint/typecheck/unit/integration → build zips → artifact store → Terraform plan/apply → smoke tests. Use Lambda aliases for canary/rollback.

# 11. Test Strategy & Acceptance

Unit: zod schemas, normalizers, adapters. Integration: S3, DDB, Step Functions (local mocks). E2E: login→upload→revalue→persist. Acceptance: SLOs met; error flows handled; rate limits respected.

# 12. Risks & Mitigations

Data source outages (mitigate with caching & circuit breakers); false authenticity flags (human-in-the-loop review); cost spikes (budgets & alerts); model drift (periodic evaluation).

# 13. Traceability (PRD → ERD)

| PRD Feature | ERD Component |
| --- | --- |
| Scan & valuation | upload\_presign + cards\_revalue + Step Functions |
| Authenticity scoring | RekognitionExtract + AuthenticityAgent (Bedrock) |
| Portfolio vault | DynamoDB schema & CRUD APIs |
| Alerts | EventBridge events + future notifications |

# Appendix A: Sample Payloads

FeatureEnvelope (output of RekognitionExtract):

{  
 "ocr": [{"text":"Charizard","conf":0.98}],   
 "borders": {"left":0.12,"right":0.11,"top":0.09,"bottom":0.10},  
 "holoVariance": 0.78,  
 "fontMetrics": {"kerning":0.92,"alignment":"centered"},  
 "quality": {"blur":0.08,"glare":0.05},  
 "imageMeta": {"width":3024,"height":4032}  
}

Bedrock prompt (AuthenticityAgent) — pseudo:

You are an authenticity expert for Pokémon TCG.   
Given FeatureEnvelope and card metadata, return:  
{ "authenticityScore": <0..1>, "rationale": "<short reason>" }

# Appendix B: Example Policies & Terraform Stubs

Terraform (API Gateway route + Lambda integration) — abbreviated:

resource "aws\_apigatewayv2\_route" "revalue" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "POST /cards/{id}/revalue"  
 target = "integrations/${aws\_apigatewayv2\_integration.revalue.id}"  
}  
resource "aws\_lambda\_function" "cards\_revalue" {  
 function\_name = "cards\_revalue"  
 handler = "dist/handlers/cards\_revalue.handler"  
 runtime = "nodejs20.x"  
 role = aws\_iam\_role.lambda\_exec.arn  
 filename = "build/cards\_revalue.zip"  
}