# CollectIQ — Engineering Execution Plan (Hackathon & Beyond)

Last Updated: October 11, 2025

Duration: 1 Week Hackathon + Post-Hackathon Scaling Vision

## 1. Overview & Objectives

CollectIQ is an AI-powered platform for Pokémon Trading Card Game (TCG) collectors.   
Its mission is to become the most intelligent and trustworthy digital vault for card collectors—enabling them to scan, identify, authenticate, and value cards using advanced AI models integrated through AWS Bedrock and Rekognition.  
  
The hackathon objective is to deliver a fully working MVP in one week. The MVP must:  
- Allow users to upload Pokémon cards (front/back)  
- Identify the card using AI  
- Estimate authenticity and compute a fair market valuation  
- Save cards into a personal vault with aggregate portfolio data  
- Run reliably on AWS using secure, scalable architecture  
  
This Execution Plan defines how the engineering team will build, test, and deliver the MVP within the hackathon week, along with a vision for post-event scaling.

## 2. System Architecture Overview

CollectIQ uses a serverless architecture designed for speed, scalability, and low maintenance.   
The system is divided into three main layers — Frontend (Next.js), Backend (Lambdas & APIs), and Infrastructure (AWS).  
  
Frontend:  
- Built with Next.js 14 (App Router), TypeScript, Tailwind, and shadcn/ui  
- Handles uploads, results display, vault management, and real-time feedback  
  
Backend:  
- Node.js-based API routes using Next.js’ serverless functions  
- Powered by AWS services: S3 for storage, DynamoDB for state, Bedrock & Rekognition for AI  
- Implements Zod for schema validation, providing strong type contracts between layers  
  
Infrastructure:  
- Deployed via AWS CDK (TypeScript)  
- Uses Cognito for authentication, S3 for presigned uploads, DynamoDB for persistence  
- CloudWatch & X-Ray for observability, EventBridge for mock alerting  
  
Communication Flow:  
1. User uploads card image (frontend → S3)  
2. Backend invokes AI models to identify and evaluate card  
3. Data normalized, scored, and stored in DynamoDB  
4. User views valuation, authenticity score, and portfolio summary in Vault

## 3. AI Core (Identification, Authenticity, Valuation)

AI is central to CollectIQ’s functionality. The MVP will rely on AWS Bedrock, Rekognition, and mock adapters for rapid development.  
  
Identification:  
- Uses Rekognition or Bedrock multimodal model to detect card name, set, and number.  
- Fallback: use preloaded metadata and hash-based matching for mock testing.  
  
Authenticity Scoring:  
- Combines visual similarity (hash), OCR text accuracy, and layout heuristics.  
- Outputs a composite authenticityScore between 0.0–1.0 with component confidences.  
  
Valuation:  
- Pulls or mocks market data (TCGPlayer/eBay) to compute price distribution.  
- Applies IQR trimming and recency weighting for outlier control.  
- Returns `{ low, median, high, compsCount, windowDays, confidence }`.

## 4. Data Model & Contracts

DynamoDB follows a single-table schema:  
  
PartitionKey (PK) / SortKey (SK):  
- PK = USER#<userId>  
- SK = CARD#<cardId>  
  
Attributes:  
- name, set, rarity, number  
- authenticityScore, idConfidence  
- valueLow, valueMedian, valueHigh, compsCount  
- createdAt, updatedAt  
  
Example (Zod Schema):  
```ts  
export const CardRecord = z.object({  
 cardId: z.string(),  
 userId: z.string(),  
 name: z.string(),  
 set: z.string(),  
 rarity: z.string().optional(),  
 authenticityScore: z.number(),  
 idConfidence: z.number(),  
 value: z.object({  
 low: z.number(),  
 median: z.number(),  
 high: z.number(),  
 compsCount: z.number(),  
 }),  
 createdAt: z.string(),  
 updatedAt: z.string(),  
});  
```

## 5. Security, IAM & Privacy

Security is enforced at every level:  
- IAM: least-privilege for Lambda and DynamoDB roles  
- S3: presigned URLs limited to 60 seconds; uploads validated for MIME type and file size  
- Cognito: JWT validation on all protected endpoints  
- Data: encrypted in transit (TLS) and at rest (S3, DDB, CloudWatch)  
- Logging: structured JSON logs with sensitive data redacted

## 6. Team Roles & Collaboration

Team of 3 engineers:  
  
Frontend Engineer:  
- Builds Next.js app, integrates upload flow, visualizes results.  
- Responsible for responsive UI, A11y, and real-time feedback components.  
  
Backend Engineer:  
- Owns all API routes, identification/valuation logic, and persistence layer.  
- Integrates Bedrock/Rekognition mocks and real adapters.  
  
DevOps Engineer:  
- Owns AWS CDK stack, IAM roles, CI/CD pipeline, CloudWatch, and deployment stability.  
- Implements cost controls and S3/DDB lifecycle policies.  
  
Workflow:  
- GitHub Flow (feature branches → PR → main)  
- PR reviews under 300 lines; merge only after lint, typecheck, and test pass  
- CI/CD via GitHub Actions deploying to Amplify or CloudFront

## 7. Detailed 7-Day Timeline

Day 1: Foundations  
- Bootstrap Next.js, Tailwind, shadcn/ui  
- Initialize AWS CDK, create S3 bucket, DynamoDB table, and Cognito user pool  
- Define Zod schemas and interfaces  
  
Day 2: Upload & Mock Identification  
- Implement S3 presigned upload and mock /identify route  
- Build frontend drag-and-drop uploader with preview  
  
Day 3: Authenticity & Valuation Logic  
- Backend: combine authenticity heuristics and pricing mocks  
- Frontend: display authenticity badge and value panel  
  
Day 4: Vault System  
- Backend: /cards list & get routes  
- Frontend: vault grid, portfolio summary chart  
  
Day 5: Alerts, Polish & Auth  
- Integrate Cognito login flow  
- Add price alerts (mocked)  
- Run A11y checks, Lighthouse ≥90  
  
Day 6: Testing & Integration  
- Unit & integration tests (Vitest)  
- API error handling, load tests with mocks  
- CloudWatch metrics and logs validation  
  
Day 7: Demo Prep  
- Rehearse demo script  
- Record backup video  
- Verify live & mock modes toggle properly

## 8. Testing & Validation

- Unit Tests: Zod schema validation, authenticity & valuation logic  
- Integration Tests: Upload→Identify→Save path using mocks  
- Accessibility: Keyboard navigation, ARIA labels, color contrast  
- Performance: Lighthouse ≥90, LCP <2.5s  
- Reliability: CloudWatch error rate <5%

## 9. Demo Readiness Checklist

✅ Live URL deployed (mock & live toggle works)  
✅ Upload → Identify → Value → Vault works end-to-end  
✅ Authentication stable with Cognito  
✅ Logs & metrics verified  
✅ UI accessible and visually consistent  
✅ Backup demo recording prepared

## 10. Post-Hackathon Scaling Vision

After the hackathon, CollectIQ will evolve into a production-grade platform.  
  
Short-Term Enhancements:  
- Replace mock adapters with real market APIs (TCGPlayer, eBay)  
- Add multi-collection support (Pokémon, Magic, Yu-Gi-Oh)  
- Introduce AI-based grading model (centering, edges, corners, surface)  
  
Infrastructure Scaling:  
- Move to regionalized Lambdas (us-east-1, us-west-2)  
- Implement CloudFront CDN with signed URLs  
- Shift to AppSync GraphQL API for real-time portfolio updates  
  
Long-Term Vision:  
- Introduce AI model fine-tuning via custom card datasets  
- Enable NFT-backed ownership proofs  
- Create social trading features and marketplace integrations

## 11. Risk Register & Mitigations

| Risk | Impact | Mitigation |  
|------|---------|------------|  
| Model latency (Bedrock) | Slow responses | Cache and mock; pre-warm Lambdas |  
| Image quality variance | Misidentification | Minimum resolution guard + confirm step |  
| Data API limits | API downtime | Circuit breaker + exponential backoff |  
| S3 CORS errors | Upload failure | Validate configs early + test presigned URLs |  
| Time constraint | Delivery risk | Lock scope by Day 3 |