Seaboard Modernization – Full AI Blueprint (Sections 1–9 + Mapping Appendix) **Comprehensive Source-to-Target Modernization Reference Document**

Seaboard Foods | AS/400 to NodeJS Transformation Program | AI Reference Edition

# Section 1 – Understanding the Legacy AS/400 Landscape

## Purpose

Establishes Seaboard’s AS/400 ecosystem baseline, including RPG logic, DDS data stores, and inter-program dependencies critical for AI-assisted modernization.

## Key Notes

* Legacy System Overview:
* 70+ RPG III/IV programs interconnected through CL commands and message queues.
* DSPF files define green-screen layouts tied to business workflows.
* Modules grouped by functional domains: Orders, Shipping, Claims, Production Planning.
* Dependency and Metadata Extraction:
* Extract PF, LF, DSPF schemas to structured JSON/CSV metadata for AI ingestion.
* Static analysis identifies subprogram dependencies via pgmref logs.
* Data queues (DTAQ) and data areas (DTAARA) mapped to target cache structures.
* Challenges and Observations:
* Monolithic coupling of UI + business logic.
* Inconsistent naming conventions and shared subroutines across 250+ call instances.

## Connection to Seaboard Architecture

Forms the data and dependency foundation for AI to determine program-module relationships and DDS-to-ORM schema mapping.

# Section 2 – Strategic Framework: Strangler Fig + CDC Coexistence

## Purpose

Defines the coexistence model allowing legacy AS/400 and NodeJS-based systems to run in parallel with real-time data synchronization.

## Key Notes

* Strangler Façade Proxy:
* Routes API requests dynamically to legacy or modern services using configurable rules.
* Implements versioned routing (v1/legacy → v2/modern).
* Acts as the single user-facing interface ensuring smooth transitions.
* Change Data Capture (CDC):
* CDC mirrors DB2/400 → PostgreSQL in near-real-time using IBM journals.
* Tools: Debezium, Qlik Replicate, Precisely Connect CDC.
* Ensures data parity (<100ms lag) and continuous synchronization.
* Audit & Recovery:
* Journal receivers store before/after states for rollback and compliance.
* CDC errors trigger alerts with checkpoint-based retries.

## Connection to Seaboard Architecture

This layer ensures zero-downtime coexistence, enabling hybrid traffic handling and phased migration across modules.

# Section 3 – Target 10-Layer NodeJS Architecture

## Purpose

Outlines modular NodeJS architecture designed for scalability, clarity, and cloud readiness.

## Key Notes

* 10-Layer Structure:
* 1. Presentation (React UI) → 2. API Gateway → 3. Controller → 4. Service → 5. Domain Model → 6. Repository → 7. ORM → 8. Database → 9. Cross-Cutting → 10. Integration.
* Separates concerns, simplifying CI/CD deployment and AI-driven code scaffolding.
* Coding Standards:
* RESTful design with DTOs and OpenAPI validation.
* Dependency Injection via InversifyJS or TypeDI.
* Logging via Winston; monitoring via OpenTelemetry; rate limiting via API Gateway policies.
* Security & Observability:
* Implements OAuth2.0/JWT auth, HTTPS/TLS 1.3, Vault-managed secrets.
* Continuous scanning via OWASP Dependency Check and Snyk.

## Connection to Seaboard Architecture

Defines the target modular blueprint that AI will follow during code generation and structure validation.

# Section 4 – Phased Migration and Data Reconciliation

## Purpose

Implements iterative migration of modules with validation checkpoints ensuring continuity and rollback safety.

## Key Notes

* Phased Module Cutover:
* Modules migrate one domain at a time using proxy-based toggling.
* Façade routing tables document current ownership (legacy vs modern).
* Data Validation:
* Automated checksum and record-level validation for each migrated table.
* CDC validation dashboards monitor replication lag and mismatches.
* Governance & Rollback:
* Rollback plan triggers re-routing to legacy endpoints within 10 minutes of issue detection.

## Connection to Seaboard Architecture

Enforces safe, controlled transitions between legacy and modern ecosystems with measurable validation points.

# Section 5 – Application Component Mapping (RPG → NodeJS)

## Purpose

Defines structural translation logic for AI to generate target NodeJS components from RPG metadata.

## Key Notes

* Core Mapping Rules:
* RPG Programs → NodeJS Controllers and Services.
* DSPF (UI Layouts) → React JSX components.
* PF/LF → ORM Models (Sequelize/Prisma).
* Indicators → Conditional State logic in React or Service Layers.
* Schema Conversion:
* DDS → Normalized PostgreSQL tables.
* RPG record arrays → JSON schema entities.

## Connection to Seaboard Architecture

Provides explicit translation schema ensuring AI-based program transformation maintains domain fidelity.

# Section 6 – Testing and Validation

## Purpose

Ensures functional, security, and performance parity between legacy and migrated applications.

## Key Notes

* Test Layers:
* Unit & Integration tests: Jest/Mocha.
* UI automation: Cypress/Playwright.
* API contract testing using OpenAPI validators.
* Performance & Security:
* Load tests benchmark throughput against AS/400 baselines.
* Security scans integrated in CI/CD pipeline.

## Connection to Seaboard Architecture

Guarantees consistent performance and business logic validation post-migration.

# Section 7 – Synon / CA 2E Integration Strategy

## Purpose

Extends modernization to Synon-generated programs using metadata-driven parsing and AI refactoring.

## Key Notes

* Migration Flow:
* Synon functions decomposed into NodeJS modules via logical model extraction.
* External exits transformed into standalone REST APIs.
* Alignment:
* Synon’s internal MVC aligns naturally with NodeJS M-S-C pattern.

## Connection to Seaboard Architecture

Unifies RPG and Synon transformation approaches for consistency and scalability.

# Section 8 – AI as a Modernization Accelerator

## Purpose

Leverages AI for dependency detection, refactoring, and validation automation.

## Key Notes

* AI Enablement:
* Performs dependency tree creation and DDS schema parsing.
* Generates NodeJS and React code with ORM scaffolds.
* Creates validation test scripts autonomously.

## Connection to Seaboard Architecture

Introduces autonomous AI-driven code and artifact generation for faster, repeatable modernization cycles.

# Section 9 – Governance and Knowledge Retention

## Purpose

Establishes process integrity, knowledge management, and quality governance across modernization lifecycle.

## Key Notes

* Governance Framework:
* Version control via GitFlow with code reviews and branching rules.
* Automated quality gates in CI/CD for linting and coverage.
* Knowledge Retention:
* Continuous documentation using Markdown + Swagger autogeneration.
* Central CoE maintains modernization standards and reusability playbooks.

## Connection to Seaboard Architecture

Ensures transparency, long-term maintainability, and repeatable modernization practices.

# Appendix – RPG to NodeJS Mapping Process

This appendix illustrates the transformation of RPG program structures into modular NodeJS equivalents for AI-guided implementation.

```mermaid  
flowchart TD  
A[RPG Program] --> B[Controller]  
B --> C[Service]  
C --> D[Repository]  
D --> E[ORM/Model]  
E --> F[(PostgreSQL DB)]  
B --> G[React JSX UI]  
C --> H[Shared Microservice]  
```

Summary:  
• Each RPG program decomposes into Controller-Service-Repository layers.  
• Shared subprograms become independent APIs to avoid redundancy.  
• DDS → ORM → PostgreSQL schema ensures one-to-one fidelity.  
• DSPF → React JSX ensures uniform UI modernization.  
• CDC ensures real-time DB2↔PostgreSQL sync during coexistence.