⚓ Drydock: The Offline Logic Studio

The Problem

1. EDI is overly complicated.
    - It relies on archaic, fragile formats that are brittle and hard to read.
2. Debugging is a nightmare.
    - Developers are stuck in a "Black Box," unable to see data flow until it crashes.
3. It is a "Necessary Evil."
    - The global supply chain runs on it. We can't escape it, so we have to fix how we work with it.
4. The AI problem
    - Generative AI writes code instantly, but often "hallucinates" subtle logic bugs that are impossible to catch without running real data.
5. Heavy Environment Overhead.
    - Setting up a full local dev environment (Docker, Java Runtimes, Middleware) is

Drydock

Drydock is a Desktop IDE specifically for building and debugging data transformation logic offline. We pull the logic "out of the water" (Production) and put it in a safe "Drydock" (Local Sandbox) where we can modify, adjust, or repair it in real time.

Drydock Capabilities

1. True Hybrid Editing (Code + Visual)
    - Write code directly in the text editor, and the blocks appear automatically.
    - Drag blocks, and the code writes itself.
    - Real coding power for pros, visual clarity for everyone.
2. Zero-Latency Debugging
    - Test logic against massive EDI files instantly in a local sandbox.
3. Production-Grade Safety
    - Built-in guardrails for Oracle and SQL mappings.
    - Ensures absolute null-safety before deployment.
4. Universal Compatibility No Vendor Lock-in

Core Features

1. **Hybrid Logic Editor** - A dual-view interface (Visual Blocks + Text DSL) with instant bi-directional sync.
2. Drydock Script (DSL) - A proprietary, concise, null-safe scripting language designed specifically for data piping.
3. Live Data Inspector - Real-time debugging panel to visualize data states (Input $\rightarrow$ Transformation $\rightarrow$ Output) without deploying.
4. **Polyglot Transpiler** - A modular engine that compiles logic into standardized Java/Groovy or Python.
5. Offline Engine - A complete local environment featuring a Mock Context Manager (for variables) and Legacy Data Parsers (for EDI/XML).

Requirements

1. Native Electron application (Windows, macOS, Linux).
2. Utilization of Node.js Child Processes to prevent UI freezing.
3. Enforced Null Safety logic in all generated code.
4. Export functions must wrap code in target-specific Boilerplate.
5. Inspector must strictly detect and display Data Types (String/Int/Date).

Drydock Architecture

1. The AST Manager
    - the JSON brain that handles all logic.
    - The Importer (Java/Python -> Docker)
2. Node.js (The Engine)
    - Logic executes in an isolated Node.js Child Process.
    - allows Testing of massive files or accidental infinite loops.
3. The transpiler
    - We don't export proprietary files.
    - We transpile the AST directly to Native Java or Python.
    - Zero lock-in for third Party systems

Sprint 1

Week 1:

- Setup Electron with TypeScript, React, and Redux.
- Define the JSON specification for the Pipeline and Statement objects (the internal data structure).

Week 2:

- Build the UI Scaffold and Structure

Week 3:

- Editor Integration with Google Blockly and Monaco Editor
- set up syntax highlighting for the Drydock Script Lan

Week 4:

- Set up the AST Manager