⚓ Drydock

The Data Station

The Lightweight IDE for Rapid Data Processing

The Problem

EDI is Overly Complicated

- Relies on archaic, fragile formats
- Brittle and hard to read

Debugging is a Nightmare

- Developers stuck in a **black box**
- Data flow invisible until cloud failure

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.

What is Drydock?

Drydock is a purpose-built **Desktop IDE** designed strictly for **data transformation logic**.

Linear Workflow

Load Data (Left) → Write Python (Center) → See Results (Right)

Key Principles

- No setup
- Real Python (For now)
- Instant execution on real files
- No uploads, no waiting

Core Features (1/2)

1. Tri-Pane Linear Interface

- Rigid, intuitive workflow
- **Input → Logic → Output**

2. File-First Input Engine

- Left pane is a **file previewer**
- Drag-and-drop raw files
- Supports CSV, EDI, JSON

3. Invisible Wrapper Engine

- System handles all file I/O
- Users write only Python transformation logic

Core Features (2/2)

4. Live Data Inspector

- Auto-detects output format
- JSON / XML highlighting & folding

5. Safe-Mode Execution

- Isolated Node.js Child Process
- Strict timeouts prevent crashes

Requirements 1–2

Cross-Platform Parity

- One Electron app
- Runs the same on Windows, macOS, and Linux

Process Isolation

- User code runs outside the UI
- App never freezes or crashes

Requirements 3–4

State Synchronization

- One shared store
- Keeps Input, Script, and Output in sync

Defensive Wrapping

- Code is auto-protected
- Errors are caught safely

Requirements 5–6

Error Visibility

- Errors are always shown
- Nothing fails silently

DOM Virtualization

- Only visible lines render
- Large files open instantly

Requirements 7–8

Environment Flexibility

- Use system Python or a virtual env
- No forced runtime

Persistence Protocol

- Auto-saves work and window state
- No setup required

Requirement 9

Keyboard-First Execution

- Run with `Ctrl + Enter`
- No mouse needed

Specification Summary

Category	Count	Description
Features	5	User-facing value drivers
Requirements	9	Strict technical constraints

Sprint 1 — The Foundation

Goal: Build a working shell that handles
Files · State · Layout

Timeline: 4 Weeks

Week 1 — The Skeleton

- Initialize the Electron project
- Build the rigid 3-pane layout
  (Left · Center · Right)
- Verify window works on:
  - Windows
  - macOS
  - Linux

Week 2 — The Brain

- Install **Monaco Editor** in all three panes
- Connect the **Redux store**
- Persist editor state while typing
- Enable **Python syntax highlighting** in the center pane

Week 3 — The Input

- Build drag-and-drop zone in the left pane
- Connect the file system bridge
- Load dragged files into the editor
- Validate large text files load without crashing

Week 4 — The Polish

- **Auto-Save:** Restore code on reopen
- **Hotkeys:** Add `Ctrl + Enter` support
- **Read-Only Mode:**
  - Lock left pane (input)
  - Lock right pane (output)

Resources & Links

- **Repository:** https://github.com/pjgneck/ASE485-Capstone/tree/main
- **Project Board:** https://nku.instructure.com/courses/87393/pages/individual-project-parker-groneck-2
- **Documentation:** https://github.com/pjgneck/ASE485-Capstone/tree/main/docs
- **Progress Page** https://nku.instructure.com/courses/87393/pages/individual-progress-parker-groneck-2