Business Intelligence Platform - Project Takeover

John AOUSSOU
Senior Engineer at GL Navigation
February 2025 - present

Last updated: May 5, 2025

Project Overview

The DX-Force project aimed to develop and maintain a business intelligence platform for tracking sales performance, customer insights, and related business metrics. Key functionalities involved collecting data from sources including Salesforce and the Datable ETL service, performing complex aggregations and processing using a hybrid approach of Django ORM and raw SQL, and visualizing key metrics through a custom web application built with Django and JavaScript.

The technical stack included Python, Django (architected into multiple distinct projects: data collection API, web frontend, webhook APIs), MySQL, JavaScript, HTML, CSS, AWS (ECR, ECS, EC2), Docker/Docker Compose, Nginx.

Role and Responsibilities

Served as senior Engineer from February 2025 to present. Immediately engaged in an intense initial phase focused on stabilizing the inherited DX-Force prototype and enabling core functionality previously blocked by critical issues.

Provided senior technical leadership and architectural direction during this stabilization phase: rapidly analyzed the inherited system, defined recovery priorities, guided the team through complex issues, and personally tackled showstopper bugs and deployment blockers to restore essential functionality.

Key Challenges Faced

Upon deeper involvement, the project presented significant challenges stemming from its complex history: originally developed by an external contractor, it had seen involvement from at least four different development teams over two years prior to our team taking ownership. Key issues included:

- **Significant Security Vulnerabilities:** Inherited critical flaws, including a single AWS IAM admin key shared among all developers.
- **Inherited Technical Debt and Complexity:** A multi-component system (multiple Django projects, APIs) plagued by pre-existing issues, architectural inconsistencies.
- **Documentation Gaps:** Severely outdated, incomplete, or incorrect documentation forced extensive reverse engineering, code tracing, and log analysis for system understanding.
- **Complete Deployment Blockage (1.5 Years):** Critical Docker/AWS infra issues (ECR, ECS, inter-service config) prevented all production deployments for 1.5 years, demanding diagnosis of subtle environment discrepancies.
- **Lack of Reliable Testing Environments:** No staging, plus inconsistent local environments, prevented reliable testing. The 1.5yr deployment blockage eliminated the only prior verification method (direct production deploys), forcing developers to 'code blind'.
- **Critical Runtime Bugs and Data Integrity Issues:** Showstopper bugs in Python/Django blocked core functionality. Examples: flawed SQL aggregation `PERCENT_RANK`) causing incorrect visualizations, and inconsistent internal identifiers (`ProgramInfo`) breaking reports entirely.

Contributions and Achievements

Despite the challenges, rapid and critical contributions were made to stabilize the project and enable basic functionality within the first two months:

- **Led Resolution of Critical Deployment Blockers:** Led the team effort (starting March 2025) to fix critical Docker/AWS infrastructure issues (ECR, ECS, config), enabling the first stable deployments since June 2023 (achieved April 2025).
- **Fixed Core Application Bugs and Data Pipelines:** Identified and resolved showstopper bugs rendering key features unusable or incorrect, ensuring accurate data processing and display:
  - Addressed critical inconsistencies in internal program identifiers (`ProgramInfo`) that previously **caused complete failure** of reporting features, enabling their functionality.
  - Corrected flawed SQL aggregation logic responsible for reports generating incorrect or nonsensical results.
- **Enabled Core Functionality:** The resolution of these deployment and application-level issues allowed **approximately 20% of the platform's features, previously blocked or entirely non-functional, to operate correctly** immediately following the stabilization effort.
- **Established Functional Development Environment:** Successfully set up working local environments (Docker, PyCharm, WorkBench), while external management deemed it likely impossible, by resolving critical configuration/connectivity issues:
  - Resolved inter-service authentication failures within the local environment by identifying and correcting JWT secret key configuration mismatches between microservices.
  - Corrected local environment data inconsistencies caused by incorrect configuration values (e.g., API URIs, client database settings) being loaded, enabling reliable local testing and development.
- **Leveraged Team Expertise:** Accelerated complex problem resolution by identifying and applying individual team members' unique strengths and domain knowledge.
- **Employed Systematic Debugging:** Used multi-service log analysis, cross-layer code tracing (JS/Django), DB querying, and incremental isolation to rapidly diagnose root causes in the complex, poorly documented system.
- **Provided Technical Guidance:** Offered senior-level guidance and insights to help team members navigate complex codebase and infrastructure challenges during stabilization.
- **Identified Improvement Areas:** Proactively identified and reported potential areas for improvement, including security best practices (related to credential handling and JWT secrets) and opportunities for AWS cost optimization.
- **Clarified Requirements:** Persistently analyzed documents and sought missing details (e.g., for Datatable queries) to enable accurate implementation despite ambiguity.