Bart Dority — Technical Portfolio Document

Overview

Bart Dority has over ten years of experience shipping live, public-facing, data-driven websites and mobile applications. His background in creative direction — including formal study at Pratt Institute — grounds his engineering work in a Bauhaus-influenced design philosophy: prioritizing simplicity, clarity, and form-follows-function thinking. This combination of design and engineering fluency allows Bart to approach UI architecture at a systems level, thinking in tokens, components, data flows, and reuse before writing a single line of code.

Grid Dynamics — Enterprise Fintech Application

Role and Scope

As Lead Front-End Engineer for the Information Reporting pod at Grid Dynamics, Bart led a team of four front-end engineers in designing and delivering a scalable, enterprise-grade financial reporting application for a major U.S. bank. The application serves hundreds of institutional clients, each managing hundreds of financial accounts representing hundreds of millions of dollars. This application is proprietary and cannot be shown publicly.

Design System and Component Architecture

Bart architected the front end using React, TypeScript, Material UI, and Sass. Rather than building one-off components, he approached the UI as a design system — defining strict TypeScript interfaces shared across both the React application and the Node.js middleware API. This type contract enabled multiple engineers to work simultaneously on complex, evolving features with confidence.

The application was organized into discrete packages with clear separation of concerns. Shared components lived in a dedicated shared package, enforcing a one-directional import chain that prevented circular dependencies and kept the codebase maintainable as it scaled.

Bart extended the Material UI component library to meet the application's specific needs. A notable example is a searchable, multi-select dropdown with accordion-style tree grouping — users could select individual items, entire groups, or search by keyword before selecting. Originally built for the accounts page, this component was quickly adopted by multiple other teams across the application.

A generic, reusable list component powered the accounts, reports, loans, and tax-documents pages — each maintained by separate teams. Bart abstracted the shared logic into a base component flexible enough to handle single-account users, users with hundreds of accounts, and emulation mode (used by QA and product teams when working directly with customers on diagnostics and feature planning).

State Management

Bart designed the MobX state management layer with multiple purpose-scoped stores: one for user account data, one for transaction reporting, one for document downloading, and one for saved report filters. The filter store was architected using maps keyed by report type, making it report-agnostic. A single store handled filtering logic for dozens of report types, and the associated UI components could be dropped into any report context without modification. This dramatically reduced code volume and delivered a consistent, predictable user experience across all report types.

Data, APIs, and Performance

The application queried complex financial reporting APIs with support for preset date ranges (last month, last quarter, year-to-date, etc.) and custom date ranges with conditional logic — up to six months for more than 20 accounts, up to one year for fewer than 20 accounts. Users could define filter criteria including keyword searches, check number ranges, lockbox account numbers, and account types. Named saved searches were supported across both transaction and full-account reporting contexts.

To solve a critical performance problem — raw API calls were taking over 20 seconds to resolve — Bart implemented a lazy-loading strategy that fetched 100 transactions per request. Pages rendered within milliseconds, and the next batch of records was pre-fetched automatically when the user scrolled past the two-thirds mark of the current set, creating a seamless, uninterrupted scrolling experience.

For reports spanning more than 20 accounts, requests were routed to a background download manager that notified users upon completion and served as a persistent report repository for future access.

All API calls were routed through a dedicated service layer. Every component handled loading and error states explicitly, displaying appropriate spinner and error UI whenever endpoints failed.

Middleware, Back-End Integration, and Tooling

The API middleware was built as a standalone Node.js application using the LoopBack framework, connecting the React front end to Java-based back-end microservices backed by Elasticsearch and Redis. Keeping the middleware as a separate layer served two purposes: it prevented back-end microservice endpoints from being exposed in the browser, and it offloaded data mapping work away from the UI, improving response times.

Bart built custom request logging into the middleware layer to help the back-end team debug microservice issues, and wrote custom AppDynamics queries enabling engineers to trace logs by `x-request-id` across the full request lifecycle.

CI/CD and Deployment Automation

The application used a comprehensive feature flag system to keep unreleased features hidden behind flags in a fast CI/CD pipeline supporting monthly production deployments. Bart managed feature flag state across each deployment cycle. He also wrote standalone Node.js CLI scripts to automate adding

and removing feature flags from the codebase and to automate the build process for the React Native mobile version of the application.

Testing

All UI components and user-flow scenarios were tested extensively using Jest and Enzyme. Before beginning development on any major feature, Bart produced UI component and data-flow architecture diagrams based on design team deliverables. These schematics gave the team a shared visual model of all the moving pieces before any code was written.

Cross-Team Collaboration

Bart worked regularly with design, product management, QA engineering, and back-end engineering teams to plan, coordinate, and roll out new features — many of which were multi-month roadmap goals. He frequently proposed improvements to UI consistency and API design, and conceived and led several cross-team codebase refactoring initiatives.

Accessibility

Bart has built publicly accessible websites meeting ARIA compliance standards. At Dority Design Works, he was commissioned by the Long-Term Care Coordinating Council of San Francisco to build their public-facing website on WordPress. The site included prominently displayed controls for switching to high-contrast mode and increasing text size, with full keyboard navigation support across all navigation links. The site content was defined with clear ARIA rules defining the banner, navigation, main, and complementary sections, and Lighthouse ratings were in the high 90s.

Dority Design Works — Freelance & Agency Work

Web Deployments and Infrastructure

Through his own studio, Dority Design Works (ddworks.org), Bart has designed, built, and deployed dozens of public-facing websites. He manages domain registration, SSL certificates, email accounts, and web hosting for a client roster spanning a shared Virtual Private Server and AWS EC2 instances. Projects typically included logo design, print marketing, and coordinated web design — with Bart handling all design and production across print and digital.

The Reclaiming Loom — Custom LMS / CMS

When a client needed a learning management system on an aggressive timeline, Bart first delivered a customized deployment of Moodle (an open-source LMS) — configuring the interface, building the

coursework, and launching in time for the first cohorts. After gathering feedback from those cohorts, the decision was made to build a custom platform from the ground up.

The custom LMS was built with Angular, Node.js, and MongoDB, and included: user registration and login, class enrollment, course material authoring tools for teachers, real-time messaging between students and instructors, community forums, and support for text, image, and video course content. This new tech stack was chosen to enhance performance and user experience overall. Using Angular allowed for more a dynamic and modern UI, including such elements as the Material UI date-picker component, and instant messaging built into the app. It also meant owning the entire codebase allowing for maximum control over the layouts - rather than trying to overlay style templates on top of legacy code.