Technical Deep Dive: Kabbage CFaaS Architecture

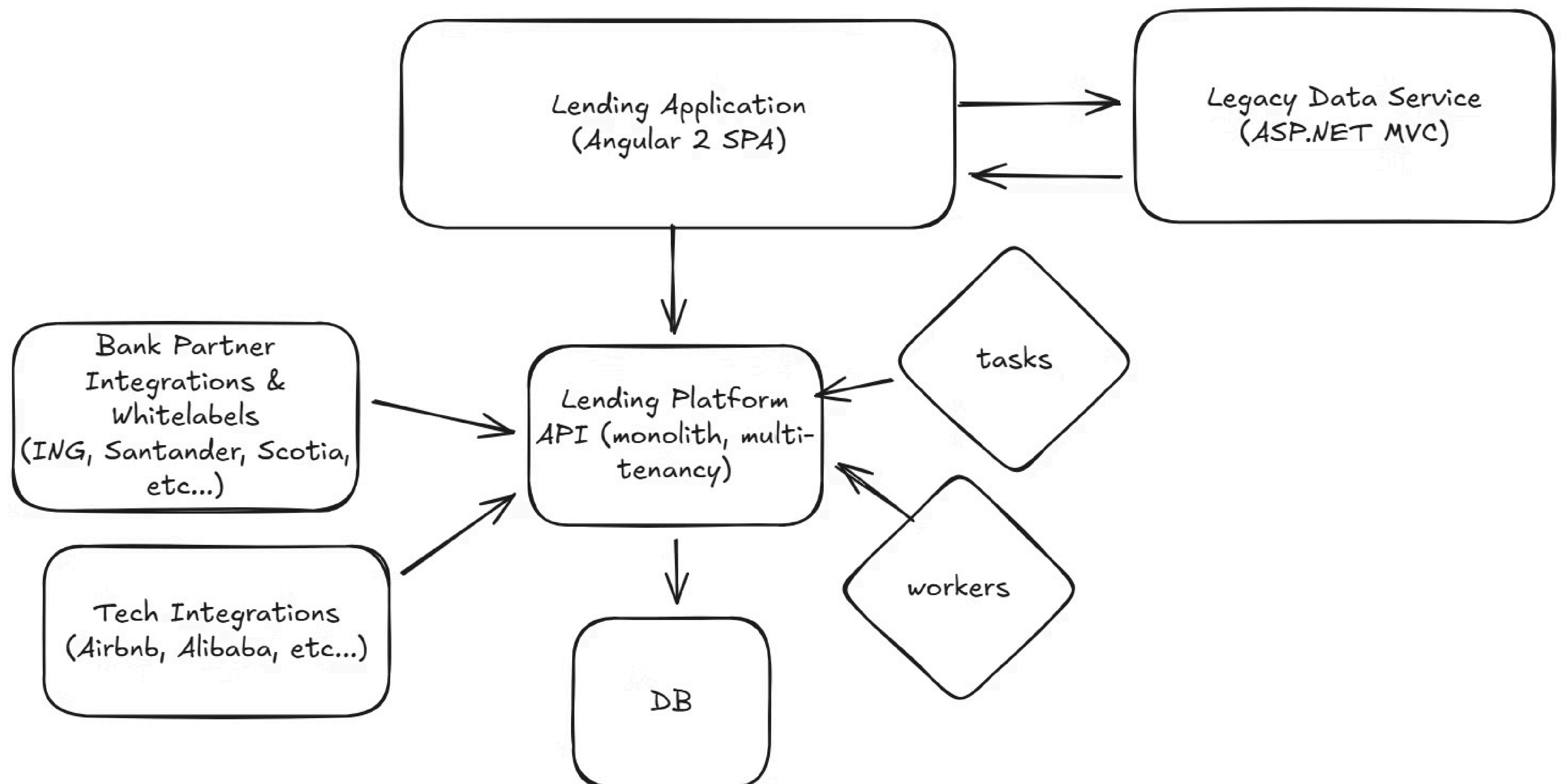
Microfrontends, Design System, and Frontend Foundations at Scale

Migrating from a legacy ecosystem to a platform-level architecture

Made with GAMMA

Before Migration: Fragmented Frontend Ecosystem

- Angular → MVC → Angular transitions created brittle boundaries
- UI inconsistencies between legacy and new surfaces
- DX friction: bloated state management, slow feature dev
- Redundant API calls, inefficient caching, perf issues

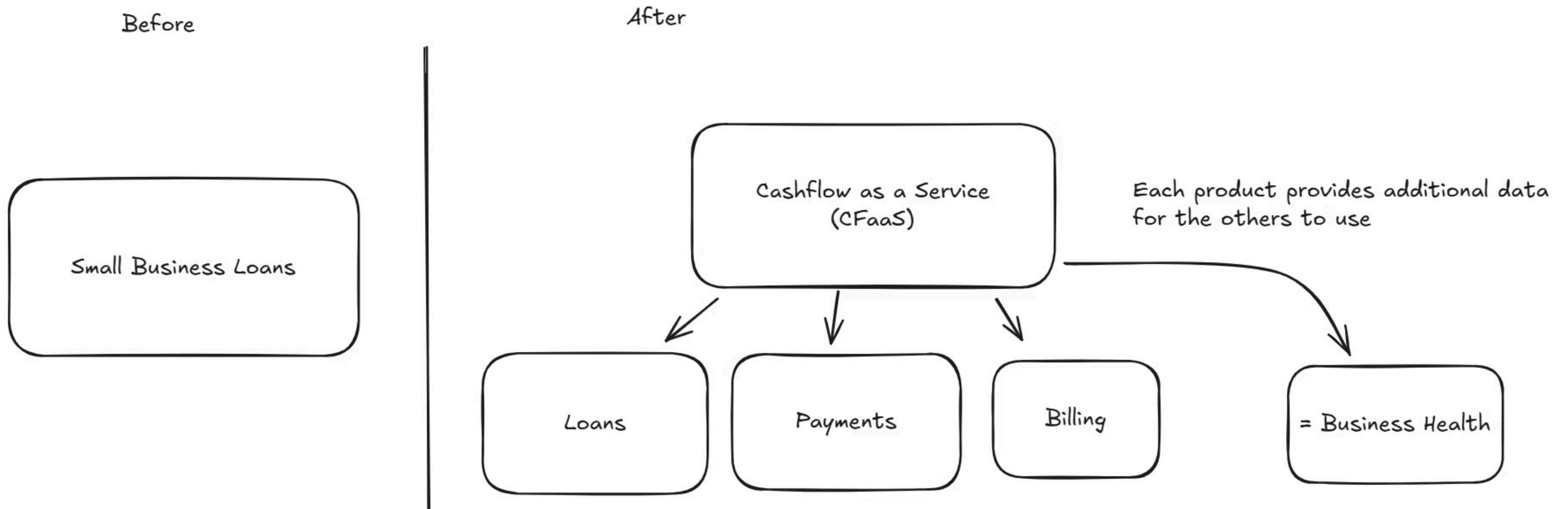


Made with GAMMA

Cashflow as a Service: A New Product Vision

Business Drivers:

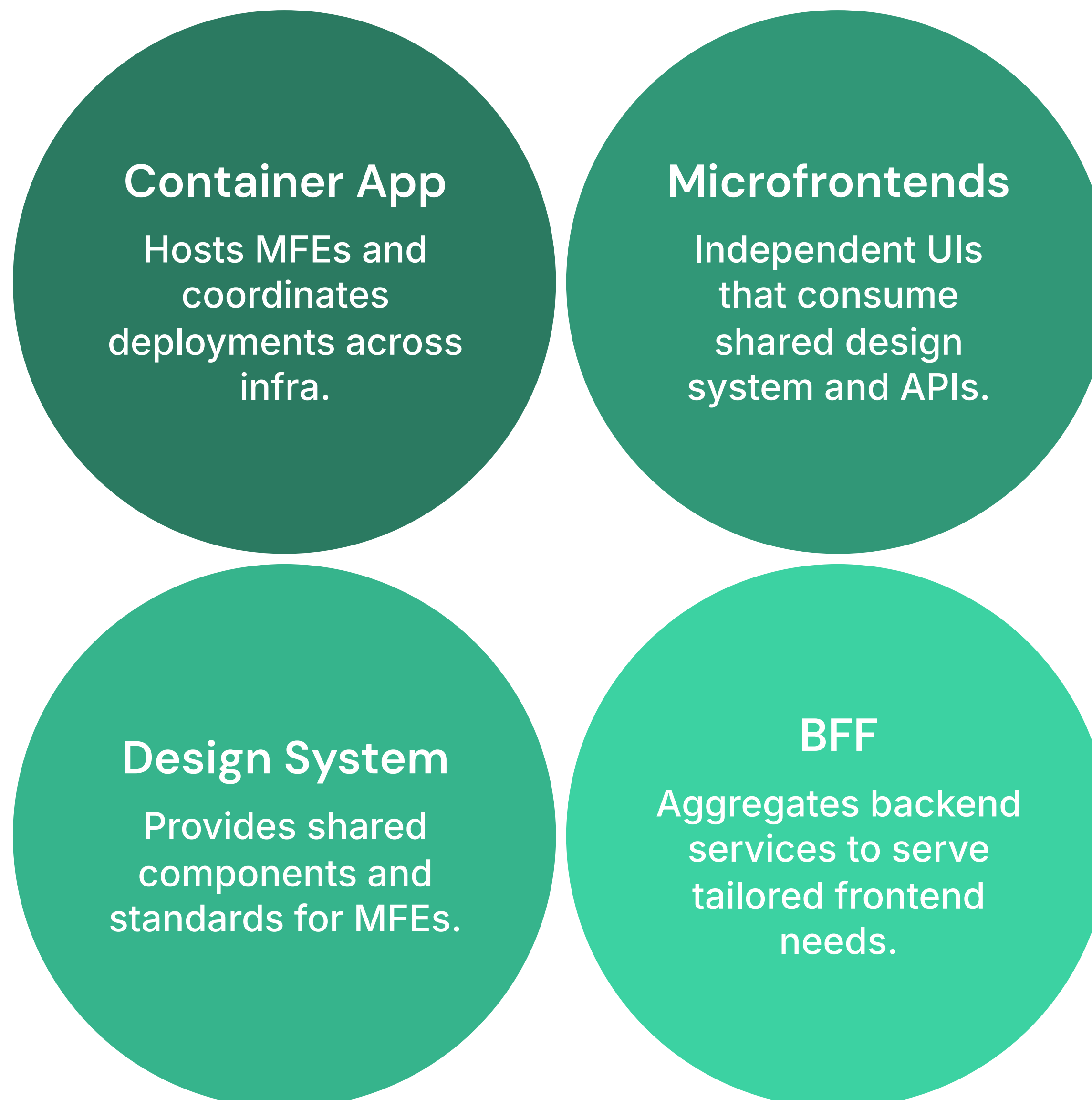
- Multi-product platform: Payments, Insights, Billing
- Independently owned & deployable apps under one shell
- Unified customer experience across products
- Faster iteration aligned with new CEO vision



Made with GAMMA

Technical Leadership

- Our team owned CFaaS core - the glue between the products.
  - *This happened to be mostly frontend*
- This eventually consisted of
  - CFaaS Container application (orchestration layer of the microfrontend),
  - CFaaS BFF (NestJS layer between the app and the API gateway)
  - Onboarding (Unified onboarding flows for all products, FE and BE).
  - Dashboard (Unified dashboard hub showing integrated view of all products)
  - Sprout Design System
  - Code sharing layer (NPM libraries)
  - Infrastructure for all of the above
- Technical lead: influenced through guardrails, code review, standards, evangelism



Made with GAMMA

Where the Old Frontend Broke Down

- Inefficient data fetching, no shared cache strategy
- Fragmented UX, missing loading/error states
- Accessibility gaps across all surfaces
- DX inconsistency limiting team autonomy
- Hard to scale features without UI drift

"Data management + architectural consistency were the biggest blockers to scaling."

Made with GAMMA

Microfrontend Implementation: Trade-offs & Best Practices

Implementing a microfrontend architecture involves strategic decisions. Understanding the trade-offs is crucial for success.

Factor	Pros	Cons
Team Autonomy	Faster delivery, independent ownership, clearer team responsibilities.	Risk of UI/UX drift, duplicated logic and effort across teams.
Tech Freedom	Teams can leverage the best framework for their specific needs (React, Vue, Angular, Svelte).	Harder hiring for diverse tech stacks, fragmented tooling, increased maintenance overhead.
Performance	Lazy load per MFE, reduced initial bundle size, better isolated performance issues.	Possible duplicate framework runtimes, overhead of multiple HTTP requests for different MFEs.
Deployment	Independent release pipelines, faster rollouts, easier rollback of individual components.	Backward-compatibility is a must, complex versioning and integration management.

Key Best Practices for Microfrontends

Adopt a Unified Design System

Ensure consistent UX/UI across all microfrontends by providing shared components and guidelines, like Sprout.

Module Federation

Efficiently share dependencies (e.g., React, Angular runtimes) and common libraries to prevent bloated bundles.

Define Clear Boundaries

Split MFEs by business domains (e.g., Payments, Insights, Billing), not technical layers, for true independence.

Automate E2E Testing

Implement comprehensive end-to-end tests (e.g., Cypress/Playwright) for cross-MFE integration and user flows.

Avoid Framework Sprawl

Limit the number of core technology stacks (e.g., 1-2 main frameworks) to manage complexity and hiring.

Contract Testing

Establish contract tests between MFEs and their APIs to prevent breaking changes and ensure stable integrations.

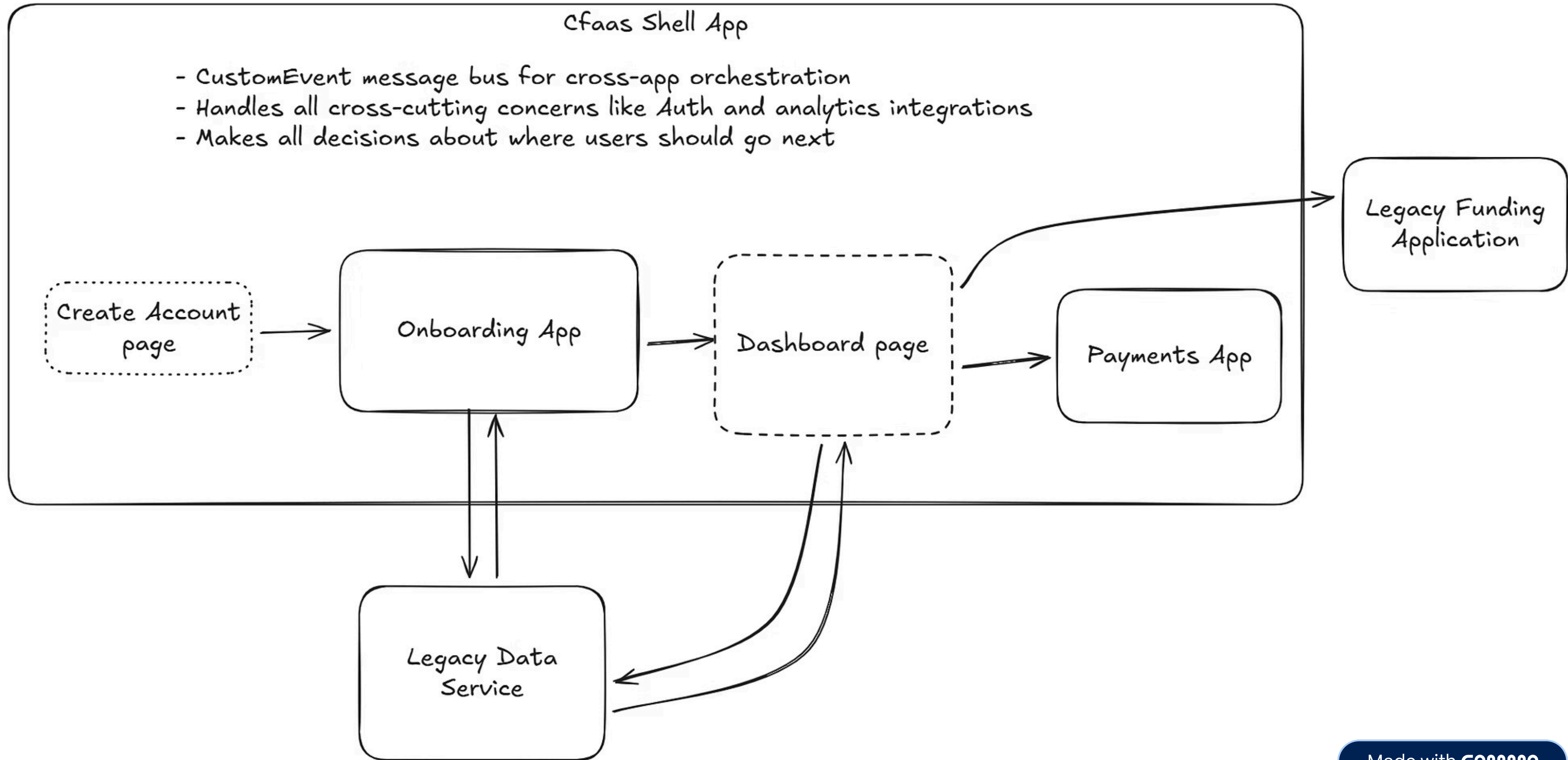
Made with GAMMA

Custom Angular-Orchestrated MFEs

- Angular Router remained the master controller
- Hybrid single-spa integration
- Custom Route Reuse Strategy to manage lifecycles
- Import-maps for shared dependencies before MF existed

Made with GAMMA

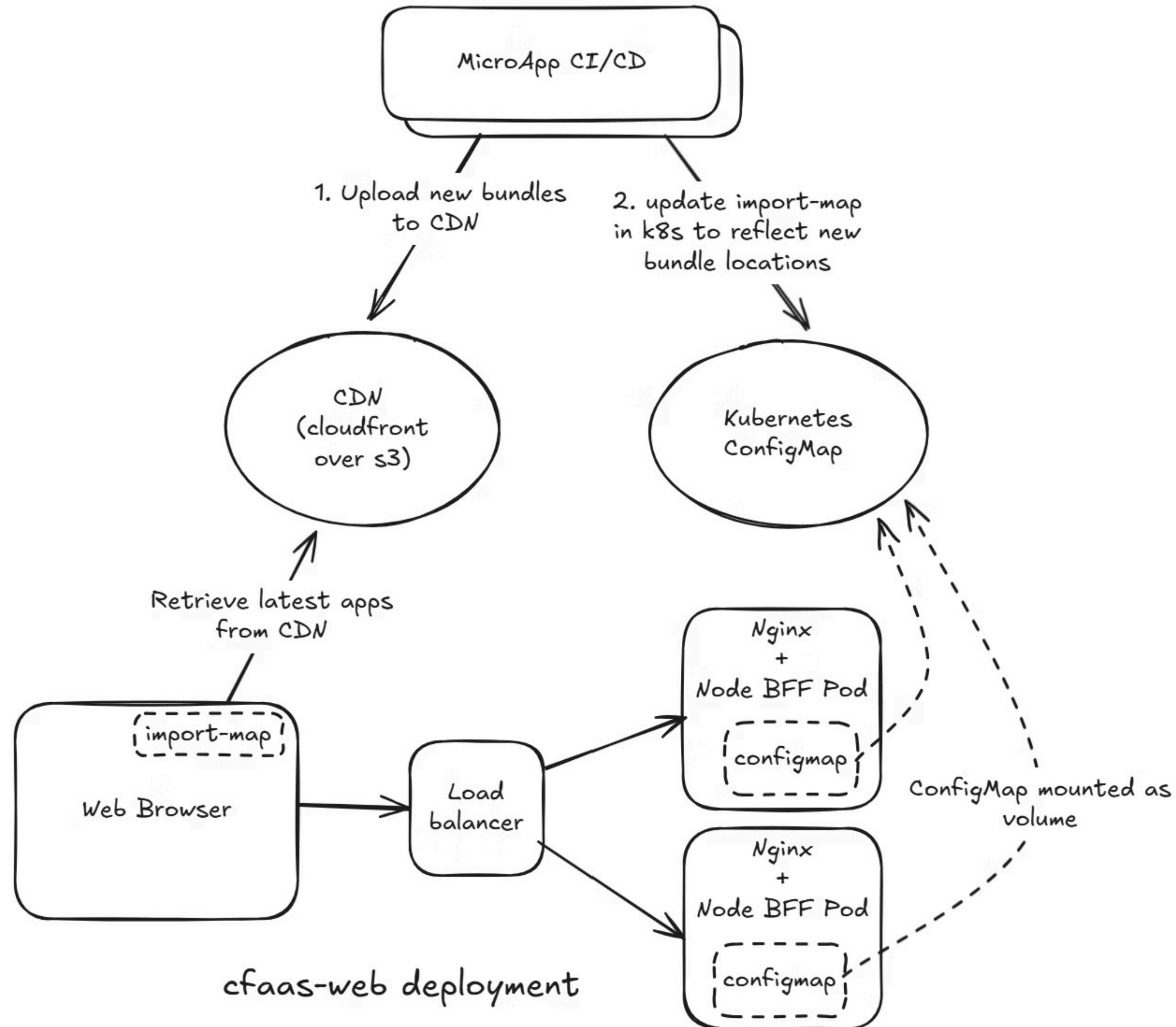
CFaaS MVP Early 2019



Made with GAMMA

CFaaS 2020

Made with GAMMA



Made with GAMMA

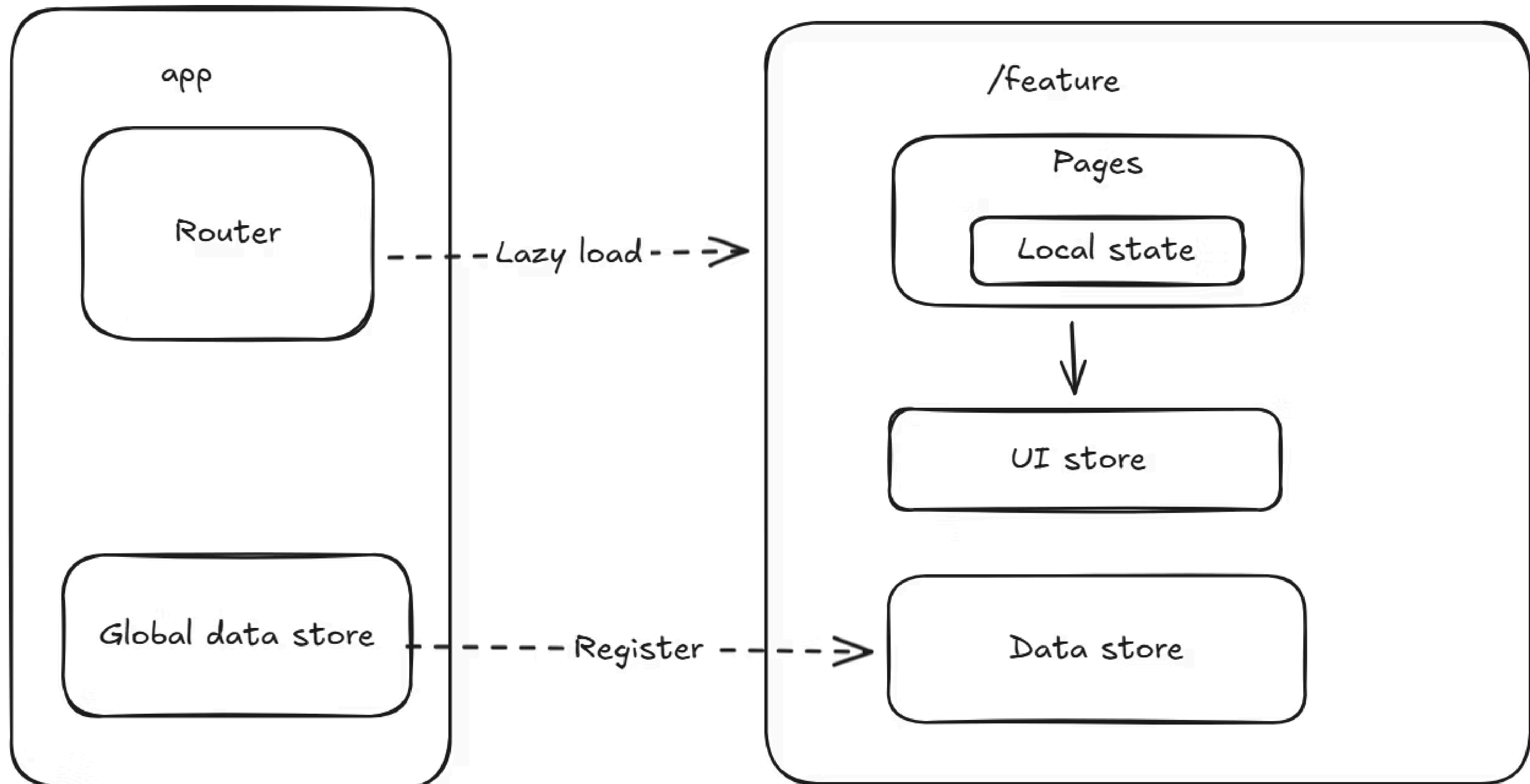
Guardrails & the "Pit of Success"

Scalable Frontend Foundations

- Strict module/library boundaries (UI, Data, Util, Domain)
- Automated MFE registration via infra config
- Unified async data abstraction (massively increased cache hit rate)
- Comprehensive automated testing suite with Cypress.
- Clear patterns for local vs global state
- Design System for improved DX and UX consistency across micro-apps.

> "Made the correct architectural choice the lowest-friction choice."

Made with GAMMA



Lazy-loaded features, scalable state

Differentiated between UI state and Data state, between local state and global state.

Used NGXS for a more expressive, less boilerplate event-based state management solution compared to the Redux-like NgRx

Features split by route, everything lazy loaded, resource hints used intelligently to preload as much as possible.

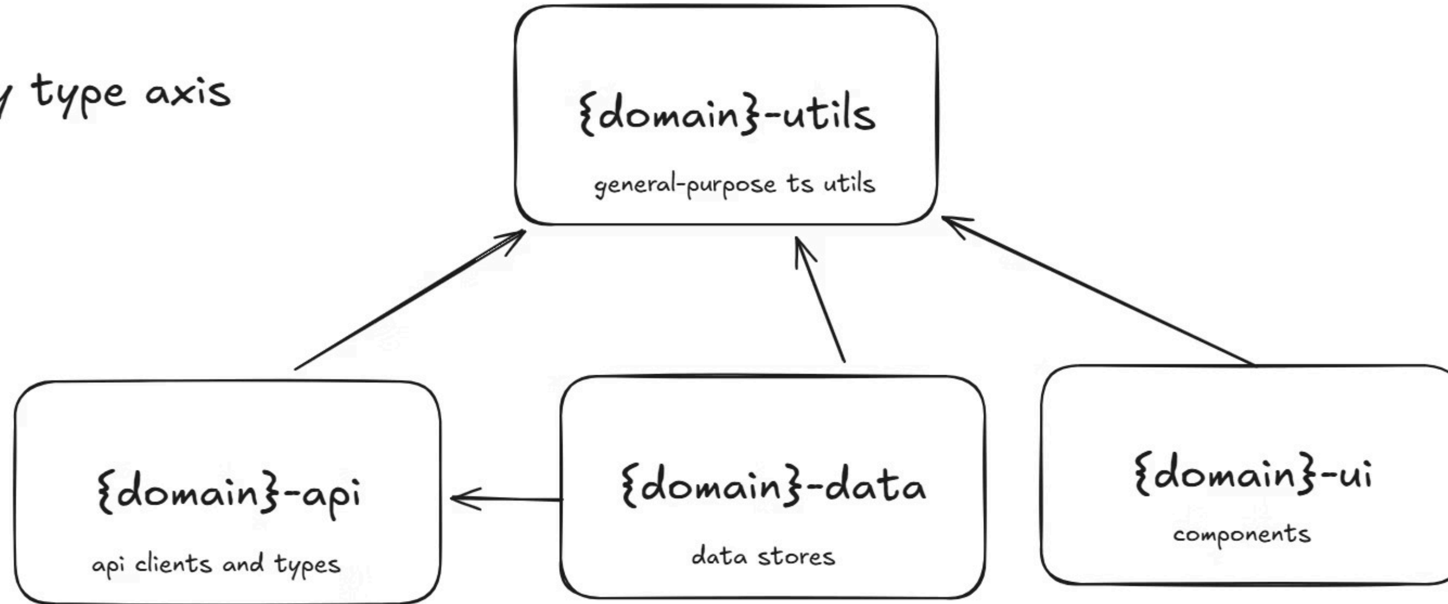
Prevent drift with automated checks, lint, and focused modules with **enforced boundaries**.

Made with GAMMA

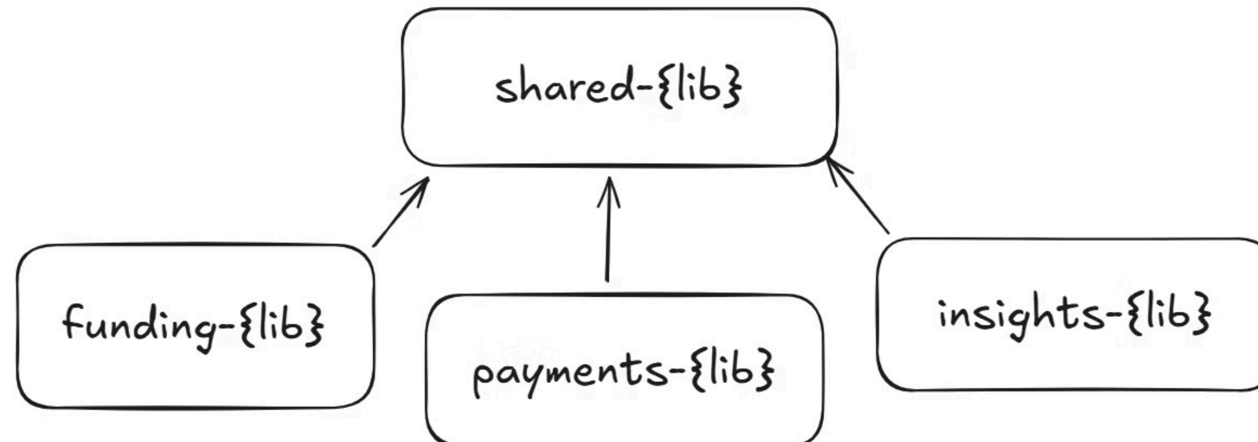
Module boundaries enforced on two axes:

Shared Libraries Architecture

1. Library type axis



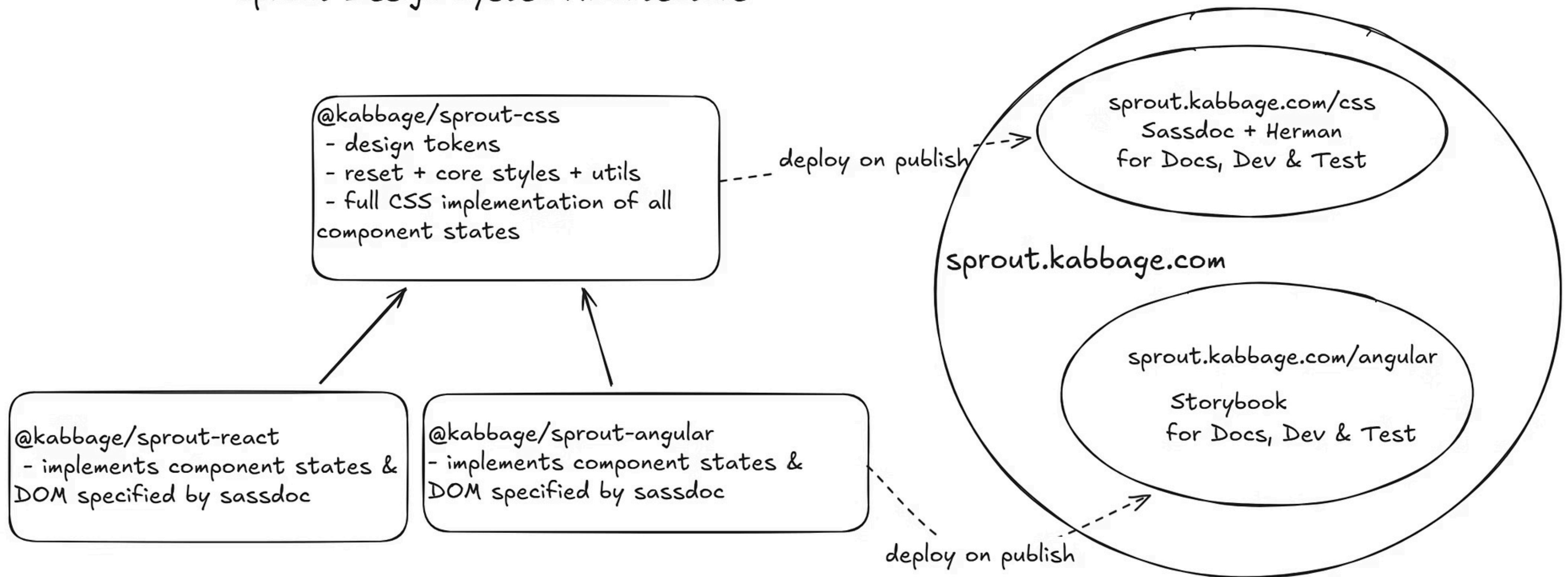
2. Domain axis



Made with GAMMA

Sprout - Cross-Framework, CSS-First System

- Core tokens (color, spacing, type, motion)
- CSS foundation shared across Angular + React
- Components documented via Herman + Storybook
- Flat class utilities for layout/visual form
- Source of truth moved fully to Storybook



Made with GAMMA

How Design, Product, and Engineering Aligned

Designers embedded per squad

DS updates flowed through feature work

Regular design-system sync

Design review as part of SDLC

Adoption spread through example + evangelism

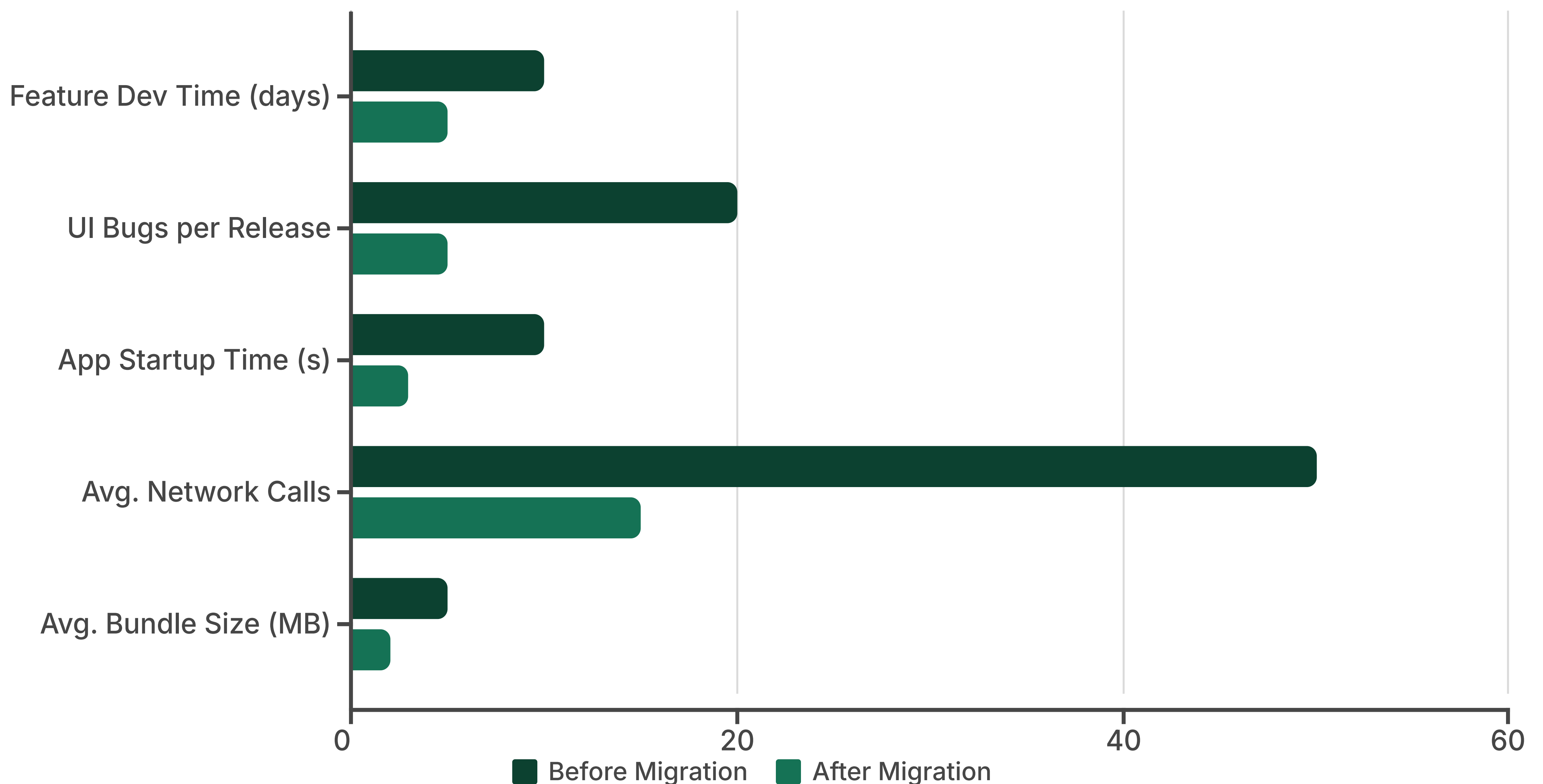
"Consistency emerged organically—patterns became easier than alternatives."

Made with GAMMA

Outcomes for Both Product & Engineering

- ~2× faster feature development
- Independent team ownership with unified foundation
- Independent deploys multiple times per day
- Major drop in UI bugs & CSS regressions
- Strong UX consistency across MFEs
- Faster startup, smaller bundles
- 60% reduction in API traffic coming from the application.

The transformation to the new architecture brought significant improvements across key metrics:



These improvements directly translated to a more efficient development process and a superior user experience.

Made with GAMMA

Lessons & Modern Improvements

- Monorepo for apps + libraries to avoid NPM bottleneck
- Rich semantic tokens + full CSS variable usage
- Component-based primitives instead of class utilities for improved DX and type-safety
- Automated a11y + visual regression testing
- Better observability into codebase patterns, UI behavior,

"Woulda coulda shoulda"

Made with GAMMA