Experiences, not Apps: Microfrontends and their BFFs

Colin Young
9/27/23

$ whoami

Consultant, 2015

MOSAIC®

Engineering Mgr, 2021

TANGIBLE

Software Eng, 2023

Backend first

Most teams that use microservices mostly just use them on the backend.

- Increase agility, decrease release times, and improve separation of concerns.
- Challenges: discoverability, documentation, enforced contracts, testing, and of course, more.
- Not perfect for all teams

Microservice transformations…
you hate to see it

How to break the monolith

- Identify domains
- Separate data
- Identify dependencies
- Make APIs

https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/

Martin Fowler

Microfrontends are microservices too.

- Separates concerns (into experiences)
- Clearly library vs. UI code
- Can be released separately(!)
- Optimize bundle size rather than giant artifacts

How to start

Define your Jobs To Be Done .
Take, for example, a lending company’s monolithic React app fed by microservices.

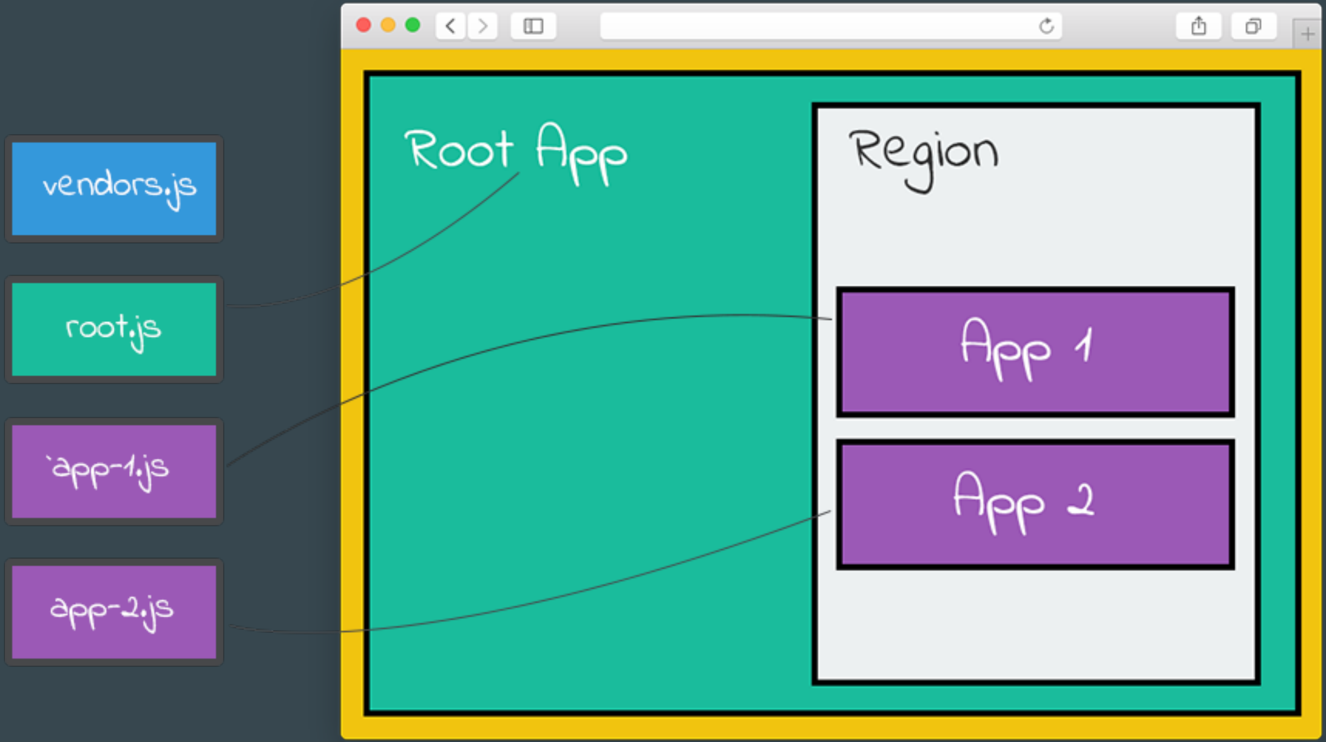
Loan Applicants	Estimate Your Payment	Apply for Credit	Sign Your Loan	Service Payments
Lead Gen Partners	Learn how to Embed Your Credit App	Customize their Co-Branding	See Referral Stats	Do Standard SaaS Stuff

Libraries

The NKOTB:
Frint.js

The Stalwart:
Single-SPA

The Minimalist:
RYO



Networking

Introduce your micros to their BFFs

Migrating to microfrontends

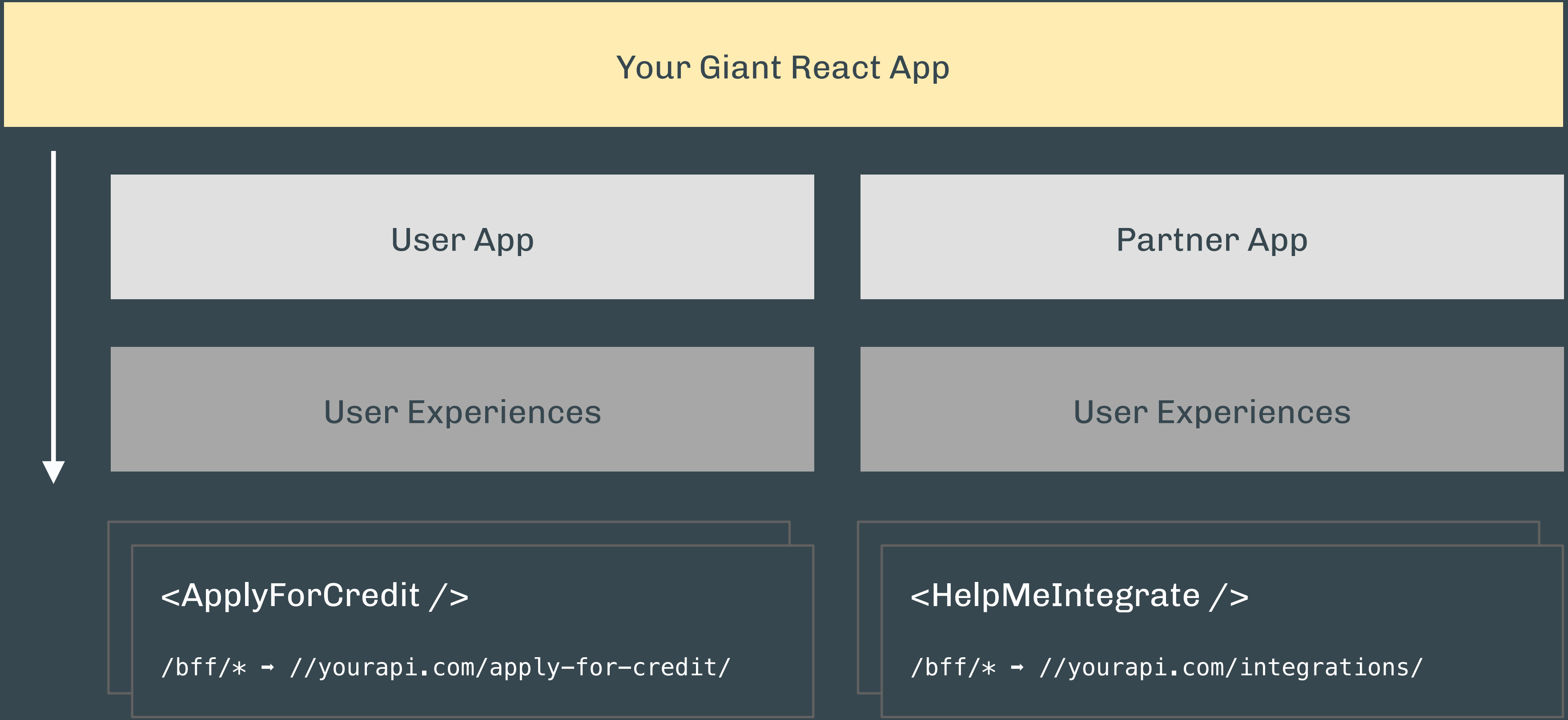
- Build on the shoulders of giants first
  - react-microfrontends
- Two types of separate apps
  - Common components
  - JTBDs with disparate business logic
- Don’t import the same thing twice
  - Import Maps
  - Tree shaking
- Divide microfrontends in a complementary way to microservices
  - Domain-based
  - If it ain’t a perfect match, introduce BFFs

Quick review

JTBDs. Lending app. Got it.

Loan Applicants	Estimate	Apply	Sign
Lead Gen Partners	Embed	Customize	Measure

Map your micros



Backends-for-frontends

Sam Newman

- Small and fast

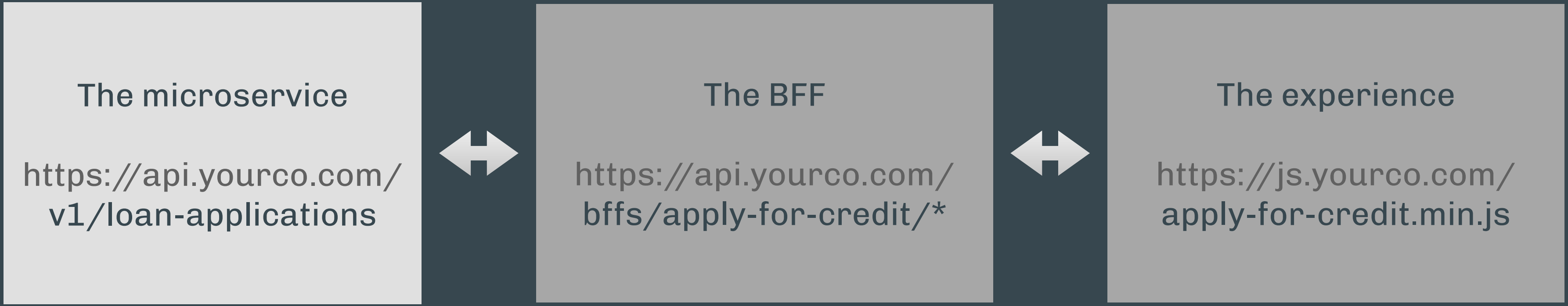
- Run wherever you want, but make it simple

- Can do the dirty work of hitting multiple services

- Basically serve two purposes:
  - Provide UI-driven types/schema
  - Safety when things change

- Just like other microservices, they can be released separately and contract-tested

BFF layout


The microservice
https://api.yourco.com/v1/loan-applications
The BFF
https://api.yourco.com/bffs/apply-for-credit/*
The experience
https://js.yourco.com/apply-for-credit.min.js

Bringing everyone together

Your team, too

Put simply

1 Identify shared chrome

2 Define experiences using jobs-to-be-done

3 Come up with schemas using product requirements

4 Make BFFs for experiences (figure out services they hit)

5 Setup release pipelines and testing

Now... Frontend can build UI without waiting on anybody

Backend knows they won't break anything

You can actually continuously deploy for the first time in your life

That’s nice, but

How do I actually do it?

Demo

Going steady

Delete the app

What do you get out of this relationship?

BE-driven schemas	→	UI-driven schemas
Gotta build dependencies	→	Shared types
Stateful	→	Stateless
Huge regressions	→	Ship separately
Single pipeline	→	Multiple pipelines
Single surface	→	Any surface

Tips to succeed

Sam Newman

- Get product folks to be advocates
- Get buy-in from devops, backend, and QA
- Use something like Storybook or the NKOTB, Ladle
- Implement contract testing (e.g. Pact)

Gotchas

Nobody’s perfect

Gotchas

- BFF RBAC
  - One mitigation: detailed JWTs
- Session
  - KISS
- Time-to-market
  - Don’t be a tiny startup
- Backend-team adaptation
  - Get frontend involved
- CSS
  - Make sure to reset and namespace

Thank you!

Resources below

Single SPA
Utility microfrontends
Module federation and System.import
Code-splitting