Experiences, not Apps: Microfrontends and their BFFs

Colin Young
2/19/25

$ whoami

github.com/colinyoung

Consultant, 2015

Engineering Mgr, 2021

Software Engineer, 2024

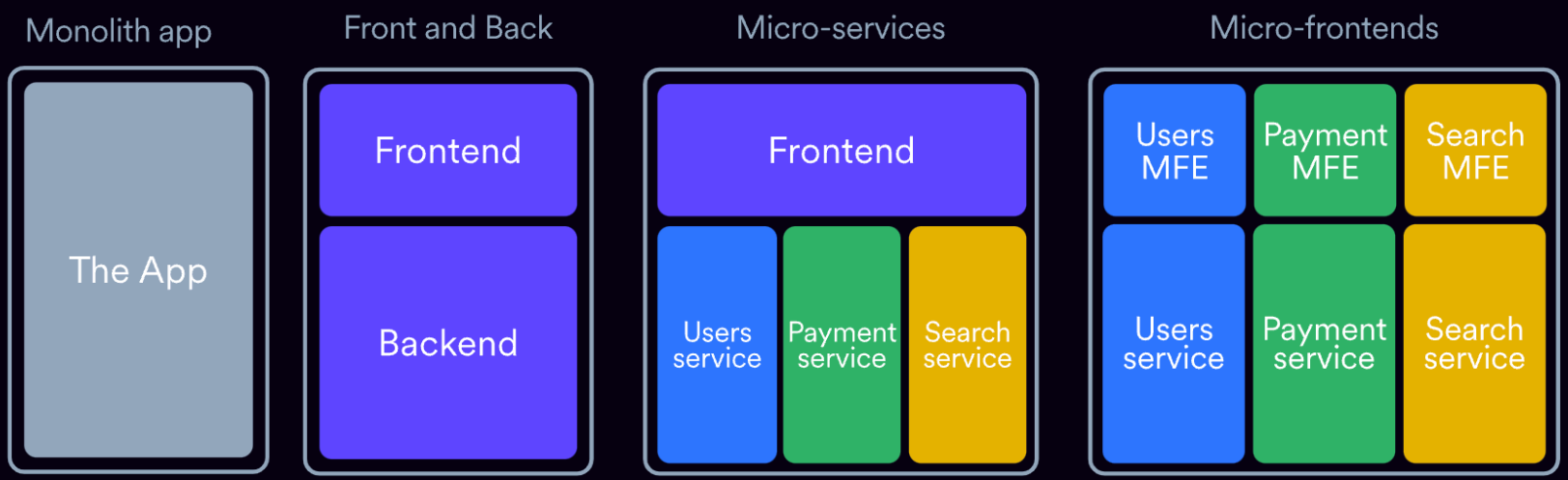
Most teams that use microservices mostly just use them on the backend.

- Promise: Increased agility, decreased release times, and improved separation of concerns.
- Opportunities: Code reuse, language-agnosticism, full-stack empowerment, and much more.
- Challenges: discoverability, documentation, enforced contracts, testing, and of course, more.

Microservice transformations…
you hate to see it

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

Martin Fowler



Evolution of composable software

Source: bit.dev

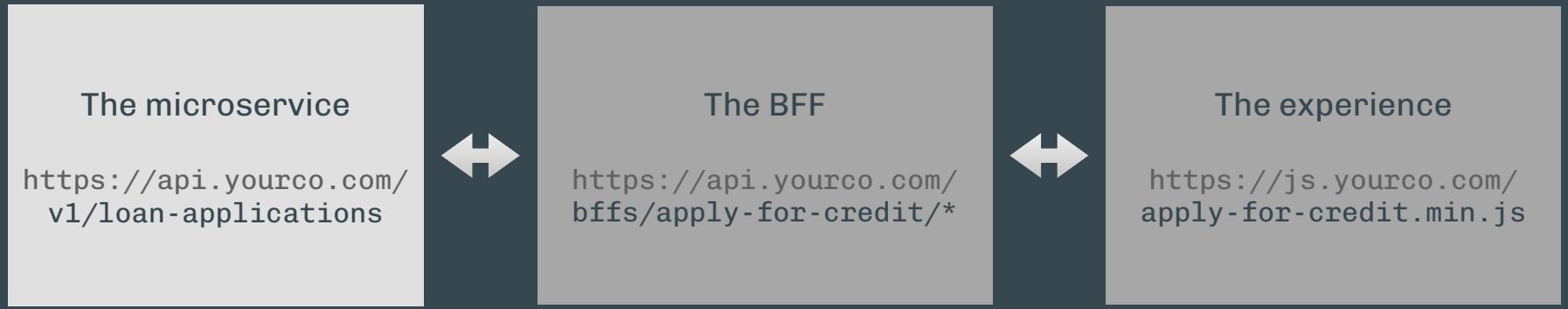
Microfrontends are microservices too.

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

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


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

In action

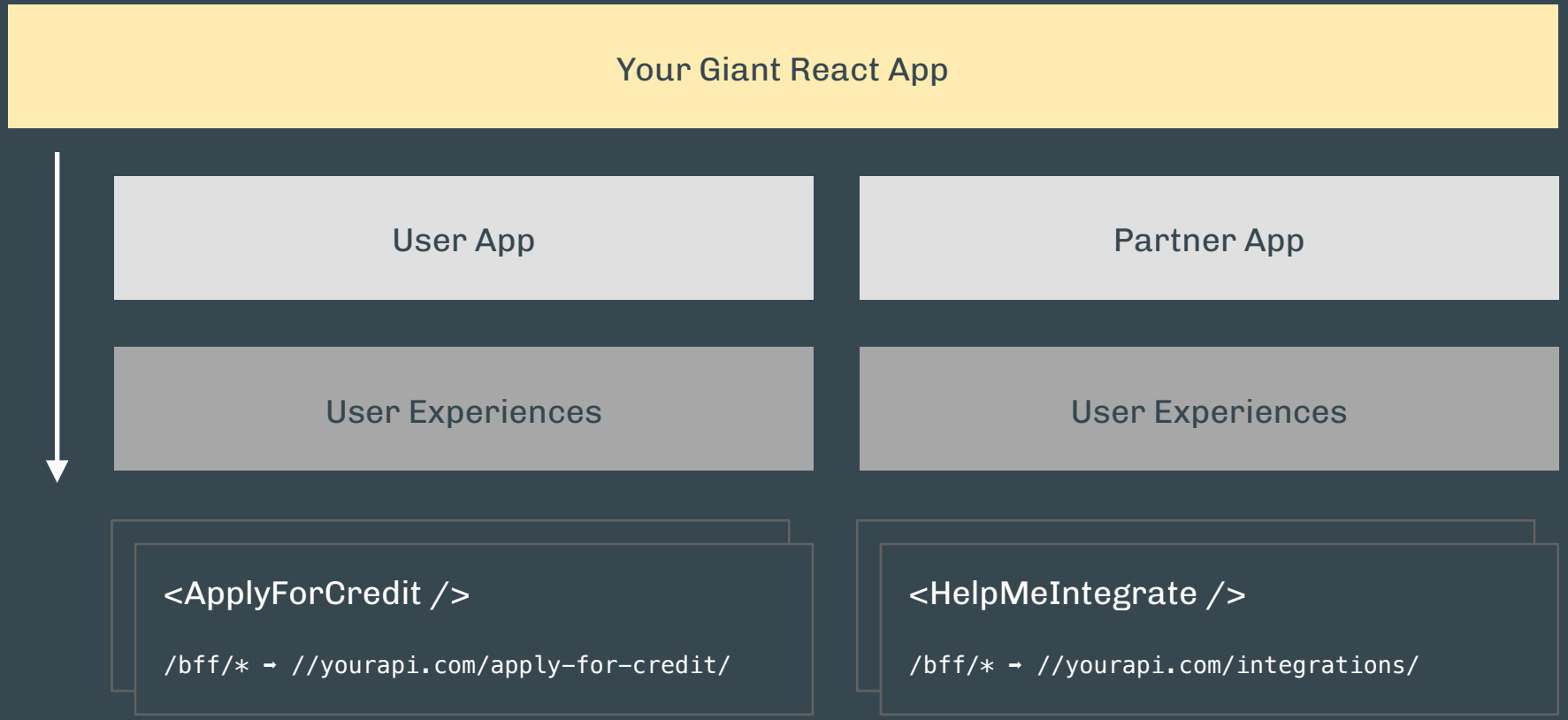
Introduce your MFEs to their BFFs

How to start

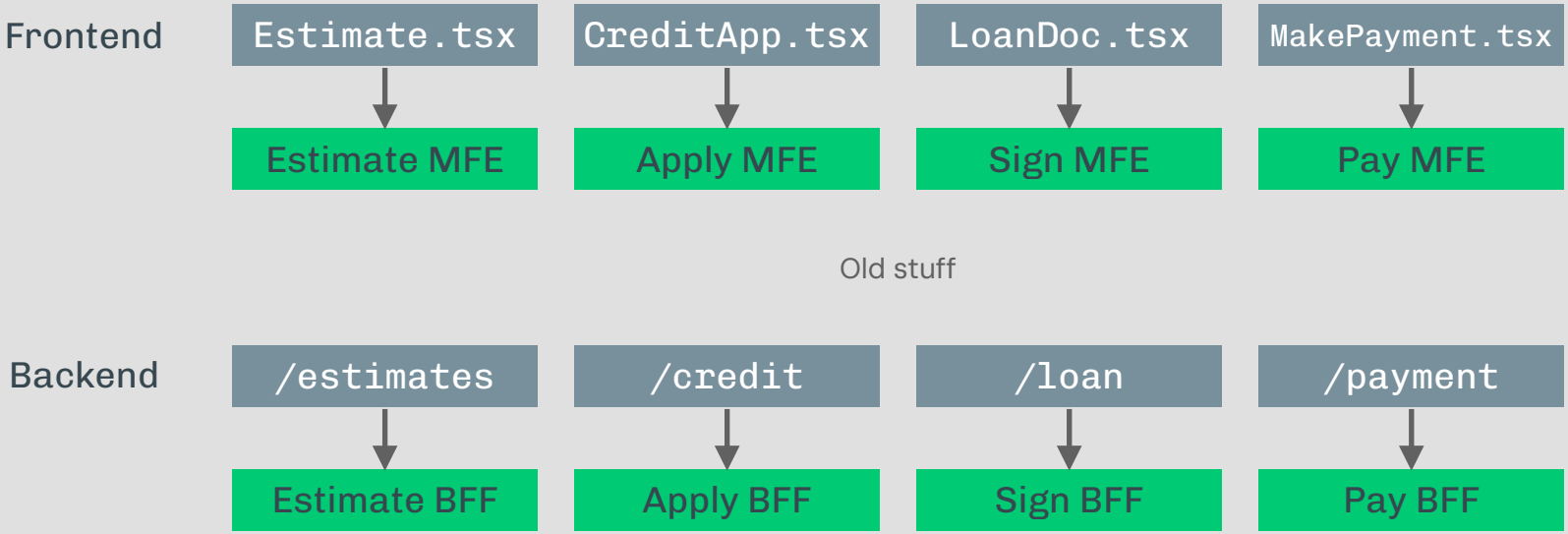
Define your Jobs To Be Done .
Take, for example, a lending company's monolithic React app fed by microservices.

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

Map your micros



Map your micros, part 2



Migrating to microfrontends

- Monorepo transformation
  - Table stakes
- Deployment options
  - Webpack Module Federation
  - Bit, Vercel MFEs, etc.
- Separation of concerns
  - Shell app + micros
  - Focus on JTBDs with disparate business logic
- Bundle optimization
  - Import Maps
  - Tree shaking
- Complementary to microservices
  - Domain-based MFEs
  - BFFs

Tips

- Think small
  - Build both MFEs and BFFs as tiny as humanly possible
  - Enables rapid iteration and comprehensive testing
- Think fast
  - How can you proxy existing services?
- Think even faster
  - Can you leverage PaaS?
  - MFEs w/ collections + functions (i.e. Firebase), not even BFFs
  - Organize by project
- Think shared
  - Design systems/UI libraries

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?

Monorepo

Source

```
packages/
  estimate/
    bff/
    mfe/
  apply/
    bff/
    mfe/
contracts/
  estimation/
    pact.json
    test.ts
  apply/
    pact.json
    test.ts
services/
  estimation-svc/
  credit-app-svc/
```

CI

```
.github/workflows
  test-pkg.yml
  test-svc.yml
  test-contracts.yml
  test.yml
```

CD

FE: Vercel
1. Frontends
2. Functions

BE: Next + Docker
Or your choice

PaaS/IAC

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
- For design, use Storybook
- Implement contract testing (e.g. Pact)

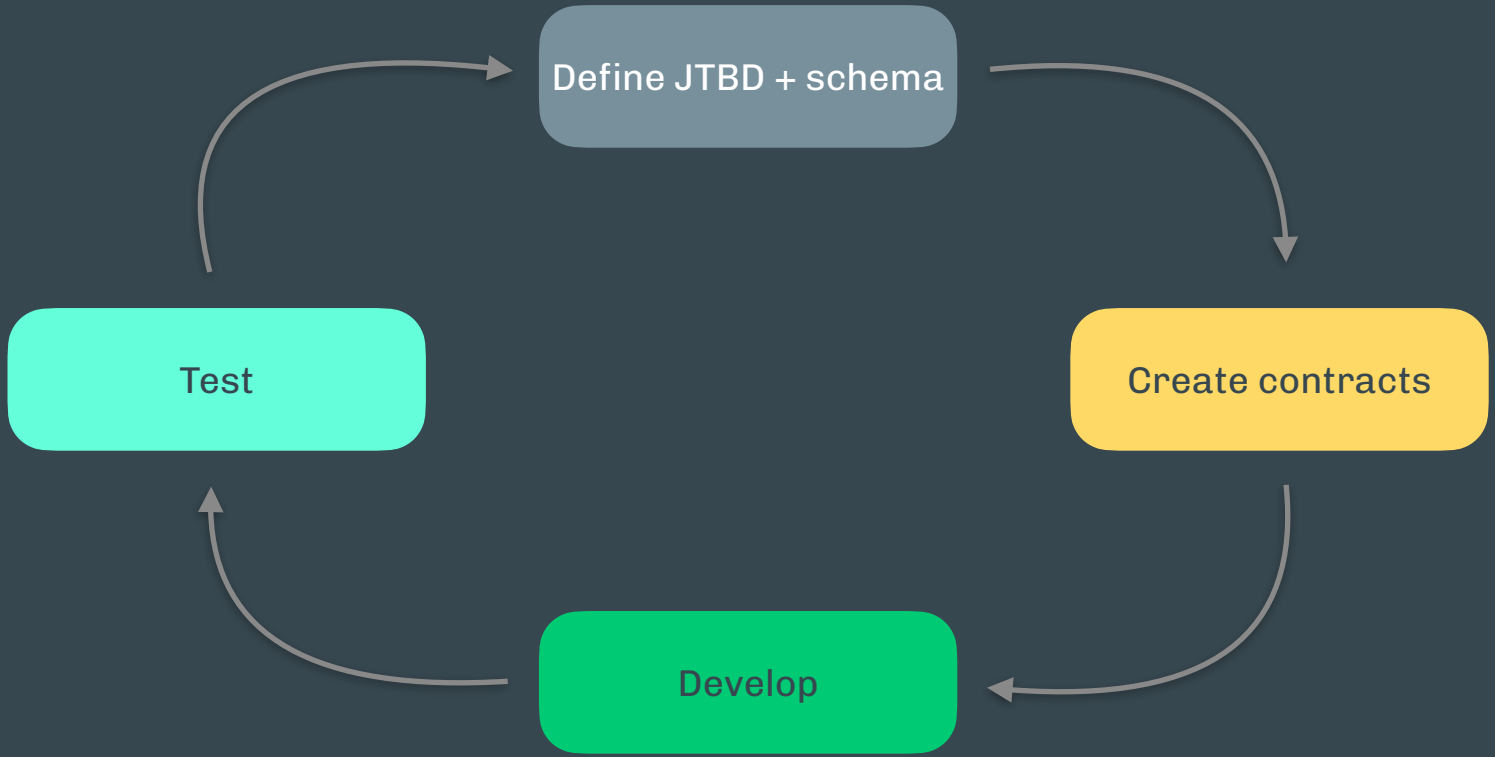
Build with AI

This is all ideally suited to AI workflows

AI works best with structure

- Contract testing → contract-driven-development
- BFFs: perfect for codegen
- MFEs with common UI: perfect for codegen
- Agentic loops, even

Iteration loop



Gotchas

Nobody’s perfect

Gotchas

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

That’s it!