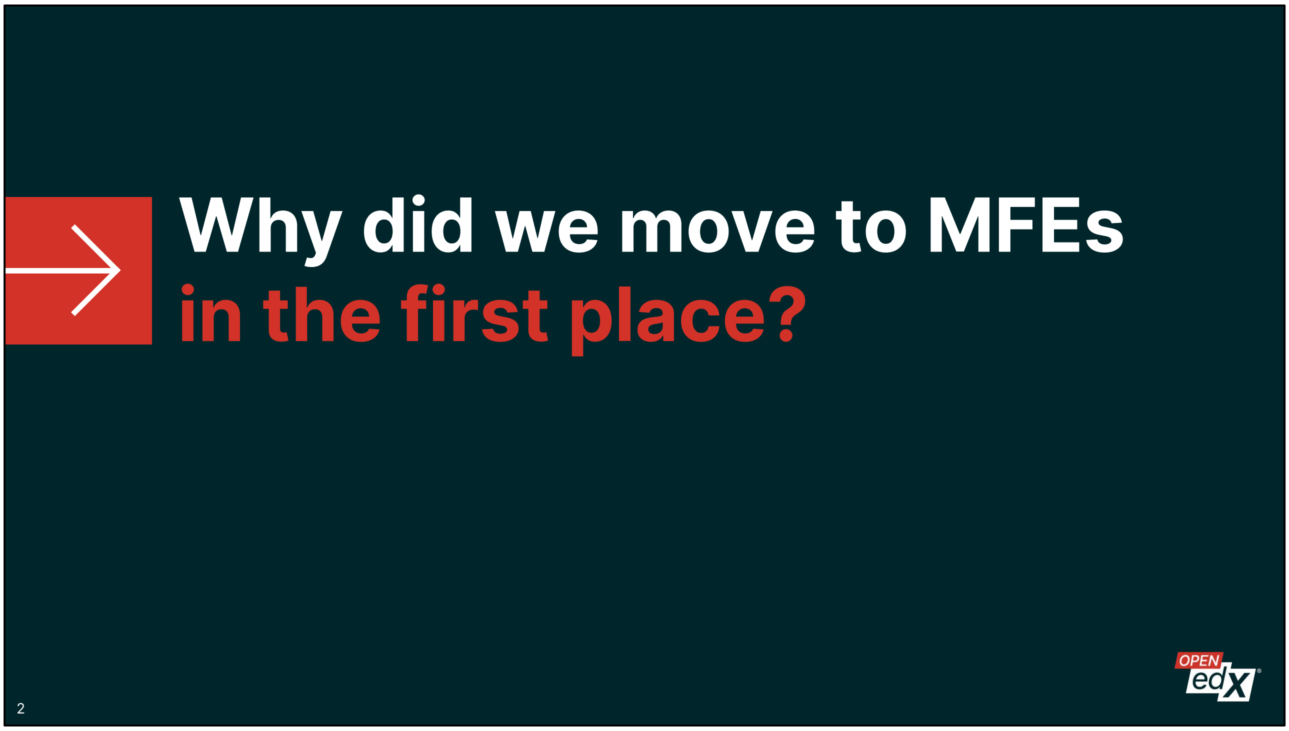
March 30, 2023

Modular Domains

Where We Want the Open edX Frontend To Go

Adolfo Brandes
Principal Frontend Engineer

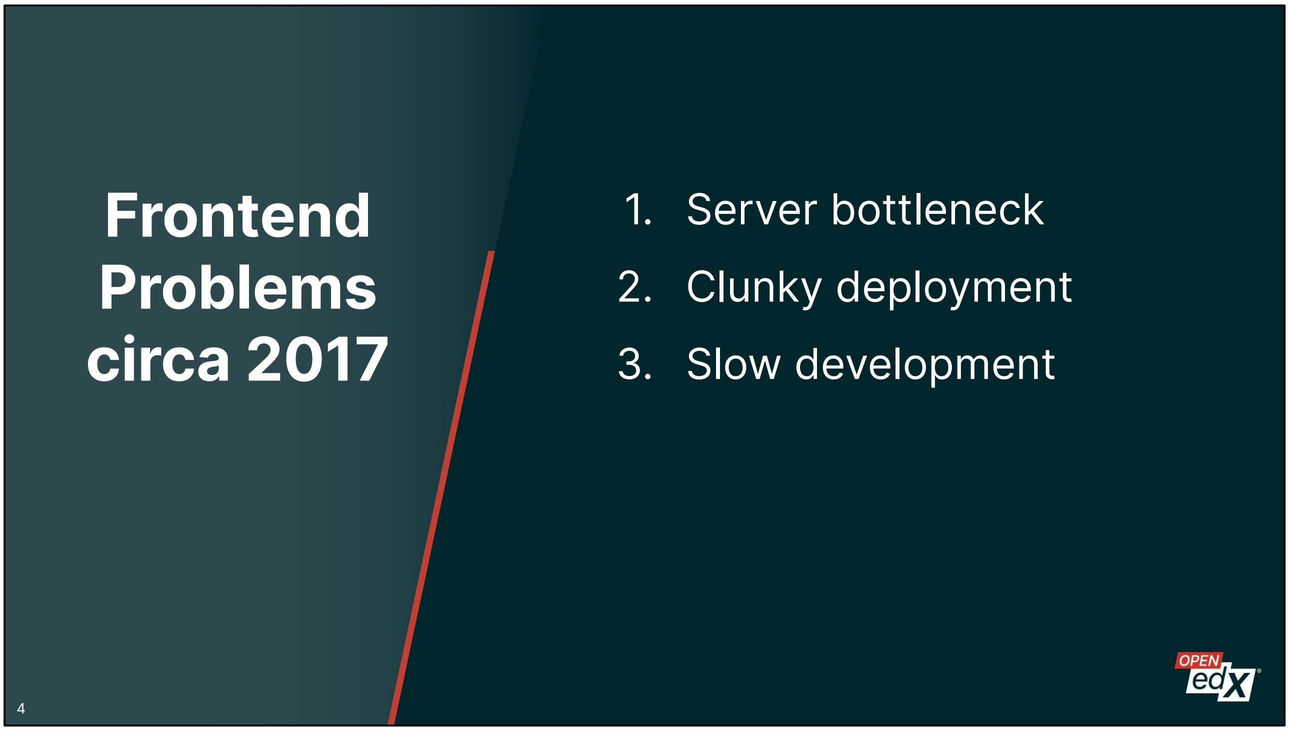


Why did we move to MFEs in the first place?

So… Why DID we move to micro-frontends in the first place? For many of us in the community, this is something that just started happening at some point, but it wasn't exactly clear *why* this was being done. Let's take a walk down memory lane…

By 2019 we started hearing noises, and in the end of 2020 we finally got a taste of them in Koa.

You all know the BOM, right? Ou affectionate nickname for edx-platform. To understand why, think about what you have to do to make the ball of mud maintain its shape.



Frontend Problems circa 2017

1. Server bottleneck
2. Clunky deployment
3. Slow development

4

Let's just make it clear that this was not something that affected just Open edX. For the most part, the project implemented the best practices of the time of its inception: in particular, those set forth by Django, the python framework chosen.

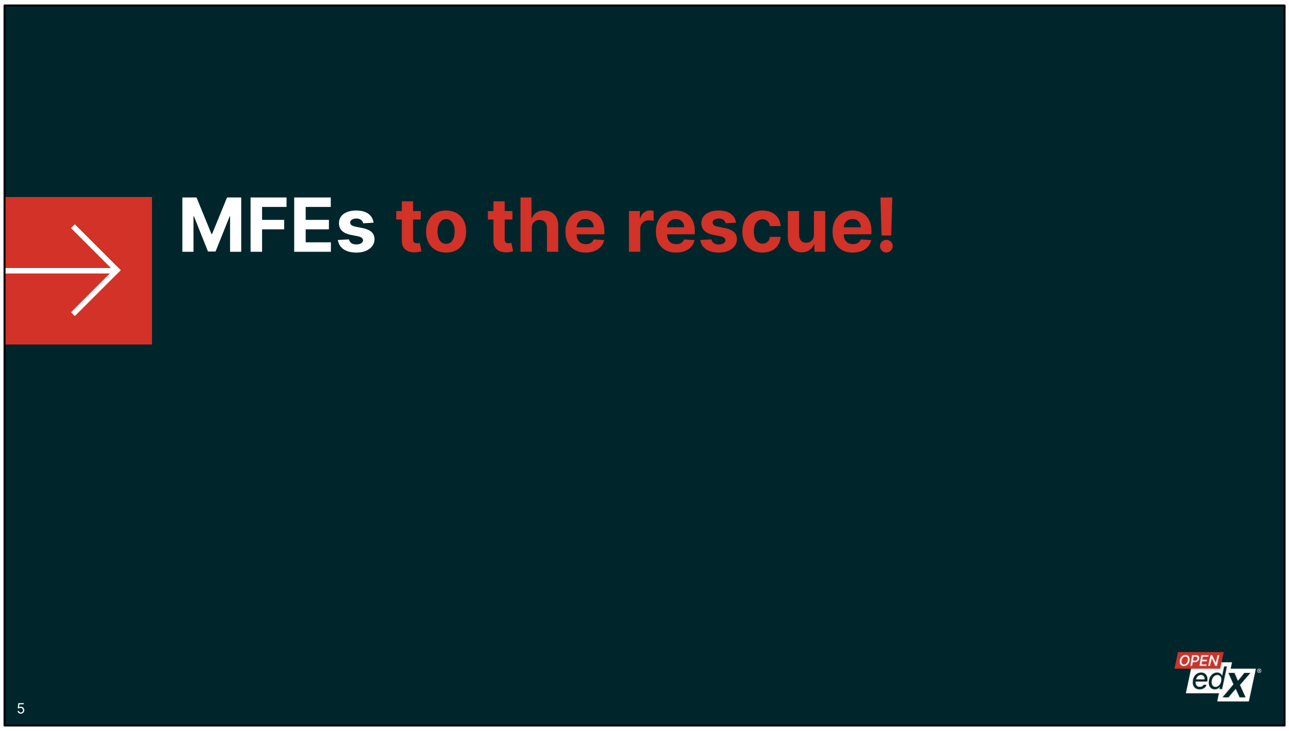
Django is great: it's a modular (remember this word!), pluggable, opinionated framework that makes it easy to not repeat oneself while developing, while also shepherding - if not dictating - the architecture into a reasonably organized and performant implementation, as was the consensus at the time. If you were just starting a project, Django would save you from reinventing the wheel until such time as it became absolutely necessary - which would usually only happen after months and years - at which time you'd be free to swap bits and pieces of boilerplate for something that worked better. Django plugins abound! (Or, you could just write your own. :shrug:)

And so, the time came when the way frontends are done needed just such an upgrade. Why? Here's a quick recap of the frontend-related problems the platform faced a few years after its inception:

1. HTML is rendered at run-time by the server. No matter how hard you cache, the server will always be a bottleneck.
2. Because you can't separate one piece of the frontend from the others - it's all one big pile - whenever you update something you have to rebuild and deploy it all at the same time.

1. This is the main problem: as the BOM got bigger, it became harder and harder to make changes: ever try to unwind a bundle of string? Coordinating the work of multiple teams became a major blocker to progress. Not to mention that properly documenting the BOM is difficult - for similar reasons

So… full rewrite then? :smirk:

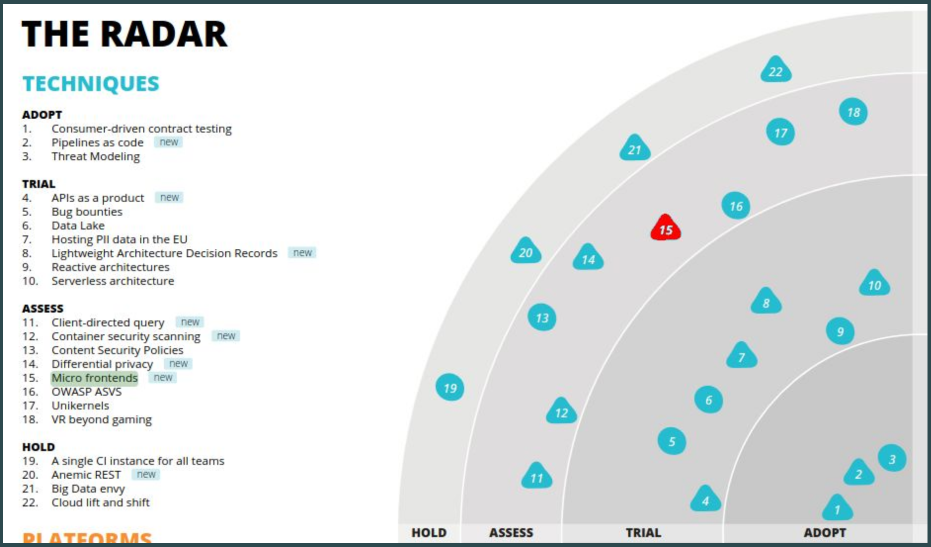


MFEs to the rescue!

Facing these problems, edx.org decided to start replatforming the frontend with something called "micro-frontends", a term that first showed up on a tech radar in 2016 (figuratively and literally).

Tech Radar Nov. 2016

First time “micro frontends” shows up.



THE RADAR

TECHNIQUES

ADOPT
1. Consumer-driven contract testing
2. Pipelines as code new
3. Threat Modeling

TRIAL
4. APIs as a product new
5. Bug bounties
6. Data Lake
7. Hosting PII data in the EU
8. Lightweight Architecture Decision Records new
9. Reactive architectures
10. Serverless architecture

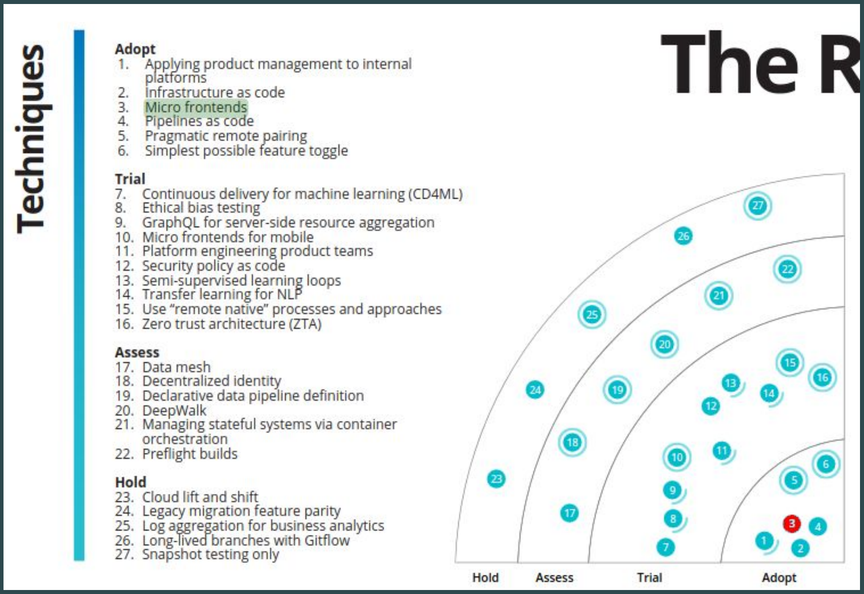
ASSESS
11. Client-directed query new
12. Container security scanning new
13. Content Security Policies
14. Differential privacy new
15. Micro frontends new
16. OWASP ASVS
17. Unikernels
18. VR beyond gaming

HOLD
19. A single CI instance for all teams
20. Anemic REST new
21. Big Data envy
22. Cloud lift and shift

The ThoughtWorks technology radar is simply a twice-yearly snapshot of a company’s perception of the technological landscape at the time. It’s not an academic exercise, but is a good reflection of the state of affairs of the industry at a particular point in time.

Tech Radar May 2020

Firmly in **Adopt** territory.



The R

Techniques

Adopt

1. Applying product management to internal platforms
2. Infrastructure as code
3. Micro frontends
4. Pipelines as code
5. Pragmatic remote pairing
6. Simplest possible feature toggle

Trial

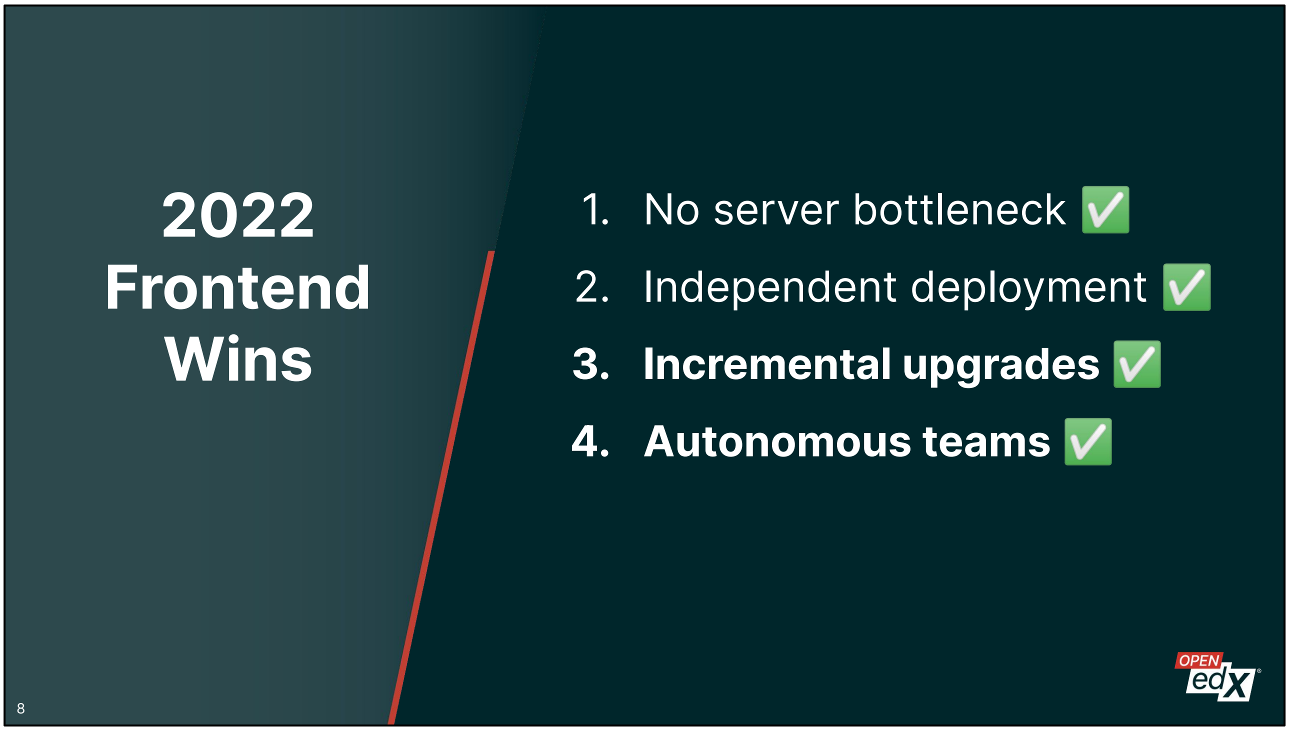
7. Continuous delivery for machine learning (CD4ML)
8. Ethical bias testing
9. GraphQL for server-side resource aggregation
10. Micro frontends for mobile
11. Platform engineering product teams
12. Security policy as code
13. Semi-supervised learning loops
14. Transfer learning for NLP
15. Use "remote native" processes and approaches
16. Zero trust architecture (ZTA)

Assess

17. Data mesh
18. Decentralized identity
19. Declarative data pipeline definition
20. DeepWalk
21. Managing stateful systems via container orchestration
22. Preflight builds

Hold

23. Cloud lift and shift
24. Legacy migration feature parity
25. Log aggregation for business analytics
26. Long-lived branches with Gitflow
27. Snapshot testing only



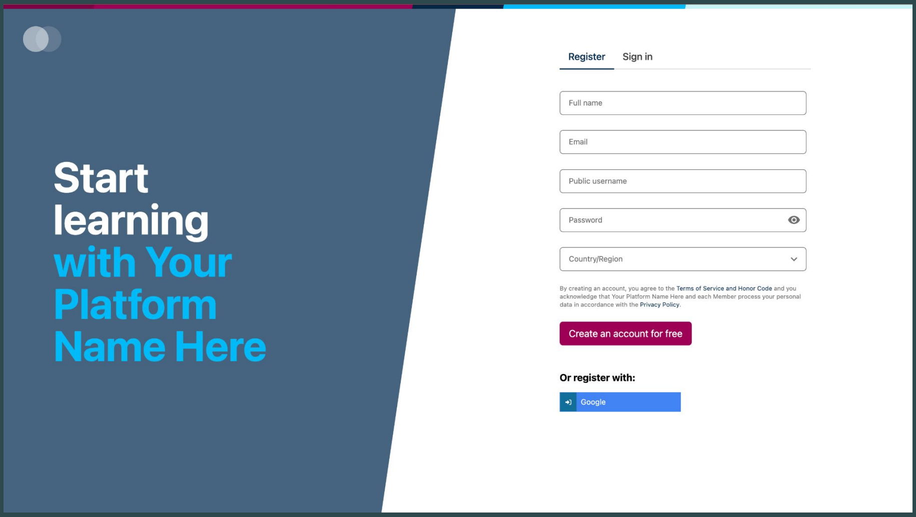
2022 Frontend Wins

1. No server bottleneck ✅
2. Independent deployment ✅
3. **Incremental upgrades** ✅
4. **Autonomous teams** ✅

So what did we get with MFEs?

1. Gone is the server request-time bottleneck for HTML. It's obviously still there for API calls - Open edX is still a centralized application, after all - but the frontend code is now delivered as fast as possible.
2. We can now deploy frontend code completely independently, with less bureaucracy and pretty much zero risk outside the MFE's scope itself.
3. Upgrading pieces of the frontend stack can now be done piece-meal, instead of having to rewrite everything at once. (Keep this one in mind.)
4. Frontend development teams can now operate pretty much independently from each other: instead of having to coordinate development across dozens (or hundreds) of people, it's now isolated to a single small team, making it much faster and efficient.

What we got!


Start learning with Your Platform Name Here

Register | Sign in

Full name

Email

Public username

Password

Country/Region

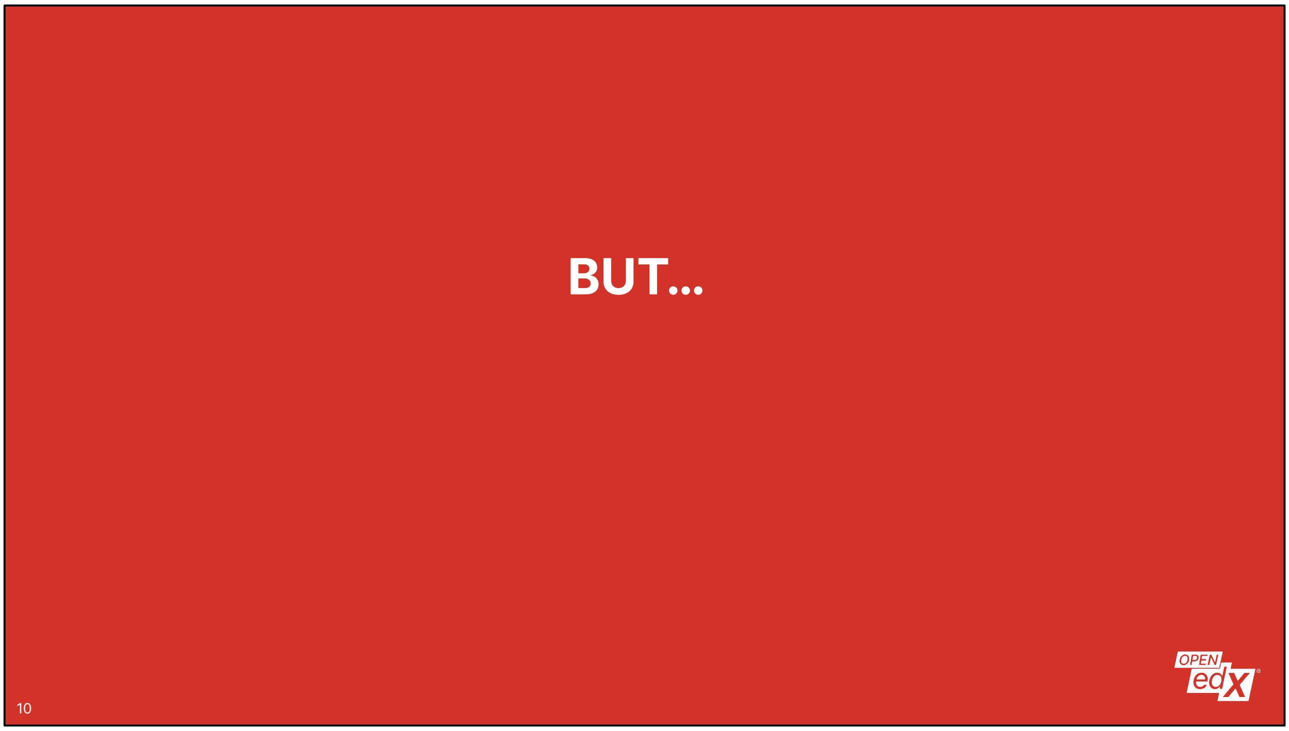
By creating an account, you agree to the Terms of Service and Honor Code and you acknowledge that Your Platform Name Here and each Member process your personal data in accordance with the Privacy Policy.

Create an account for free

Or register with:

Google

And in practice, this got us in the past couple of years, among other things: a revamped learning experience, a new discussions interface (released in Olive), a new gradebook, new authentication page… All of which would probably not have been feasible - at least not as fast - without MFEs.



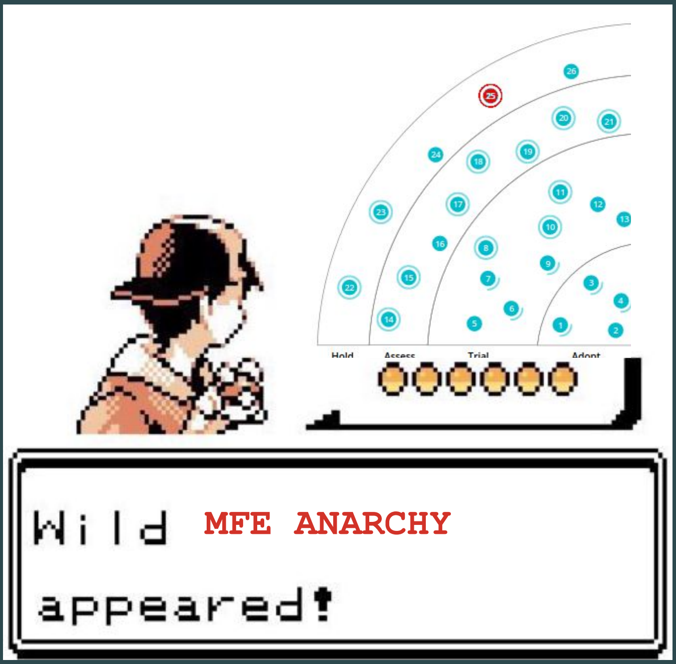
BUT…

10

There’s always a but.

Tech Radar Oct. 2020

Micro frontend anarchy: proceed with caution!



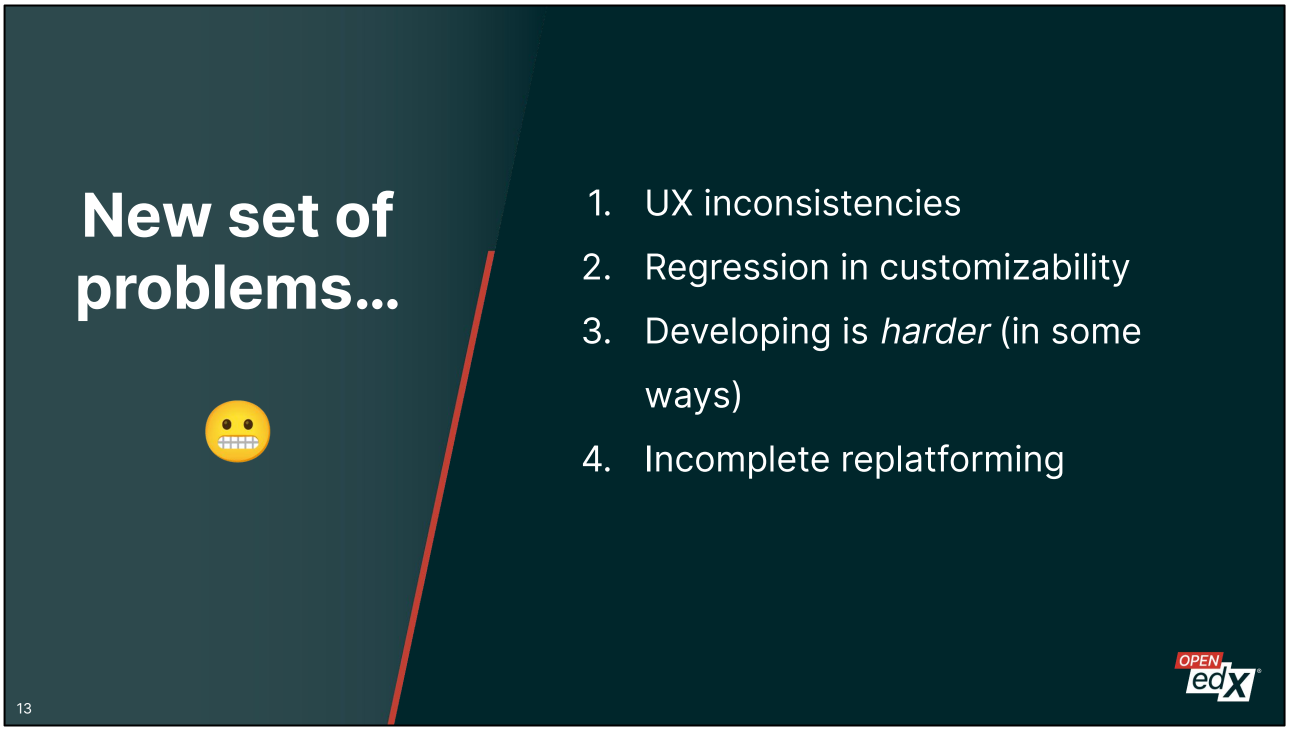
They say:

“the tendency to use this architecture as an excuse to mix a range of competing technologies, tools or frameworks in a single page, leading to micro frontend anarchy.”

Remember: this is brought up as an industry-wide concern. Another way of saying this is: MFEs are still cool, but not if you make a mess!

Completely independent micro-frontends

- Each page is a reload
- Zero interaction between MFEs
- (Somewhat) shared components and libraries
- **Complete development freedom**



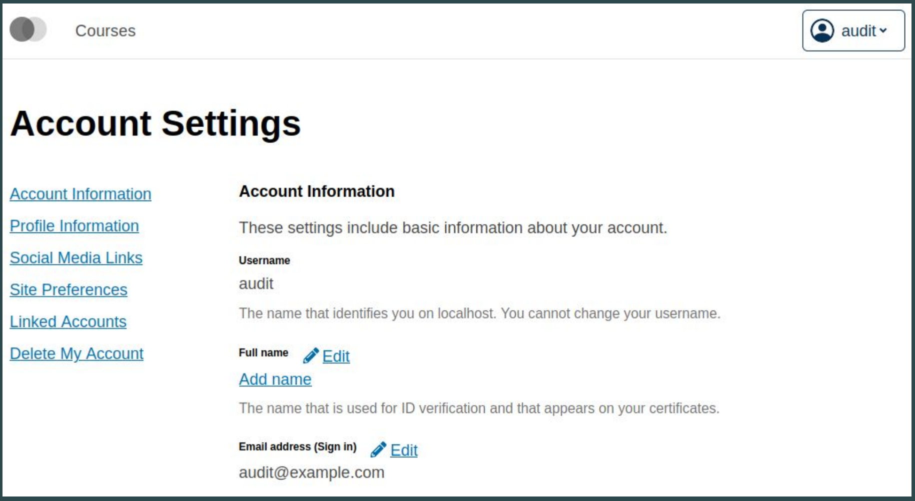
New set of problems...

😬

1. UX inconsistencies
2. Regression in customizability
3. Developing is *harder* (in some ways)
4. Incomplete replatforming

We went with the first option. And as we learned later, it has caused us some headaches. A bit of an *anarchy* even, you might say.

(About those inconsistencies..)



Account Settings

- Account Information
- Profile Information
- Social Media Links
- Site Preferences
- Linked Accounts
- Delete My Account

Account Information

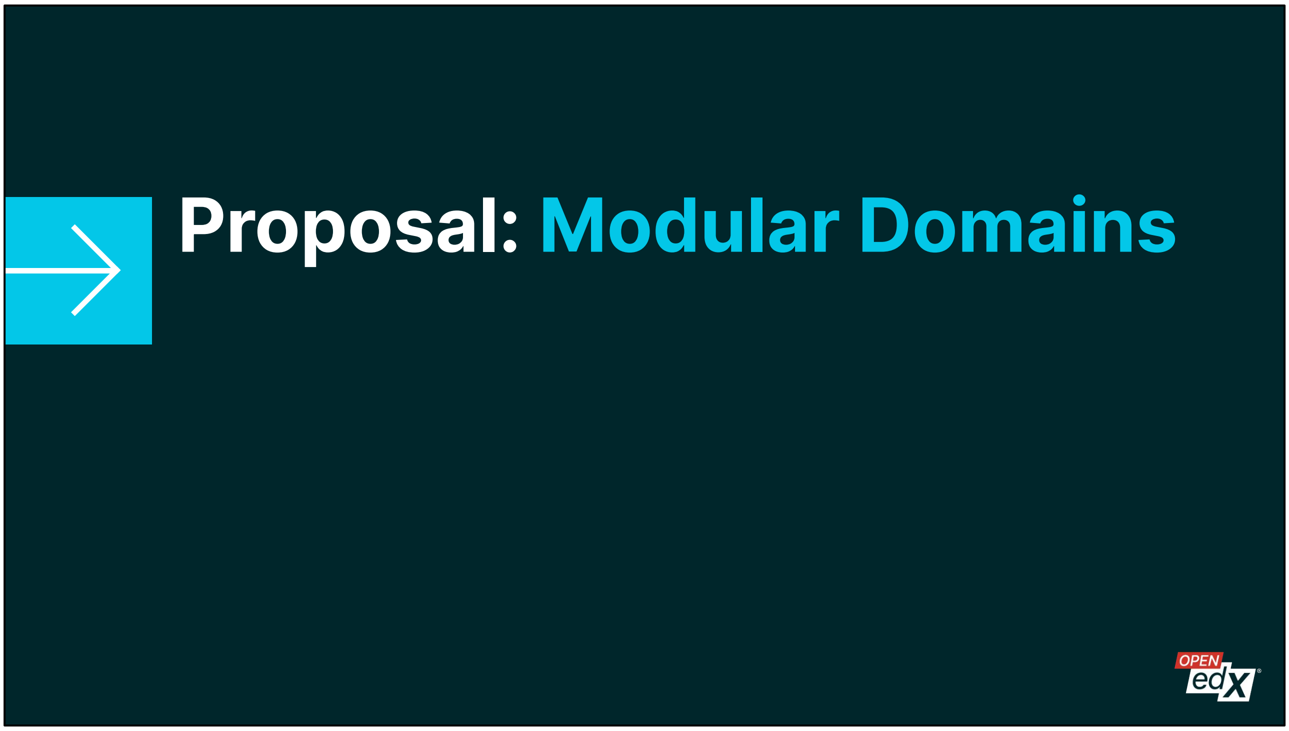
These settings include basic information about your account.

Username
audit
The name that identifies you on localhost. You cannot change your username.

Full name Edit
Add name
The name that is used for ID verification and that appears on your certificates.

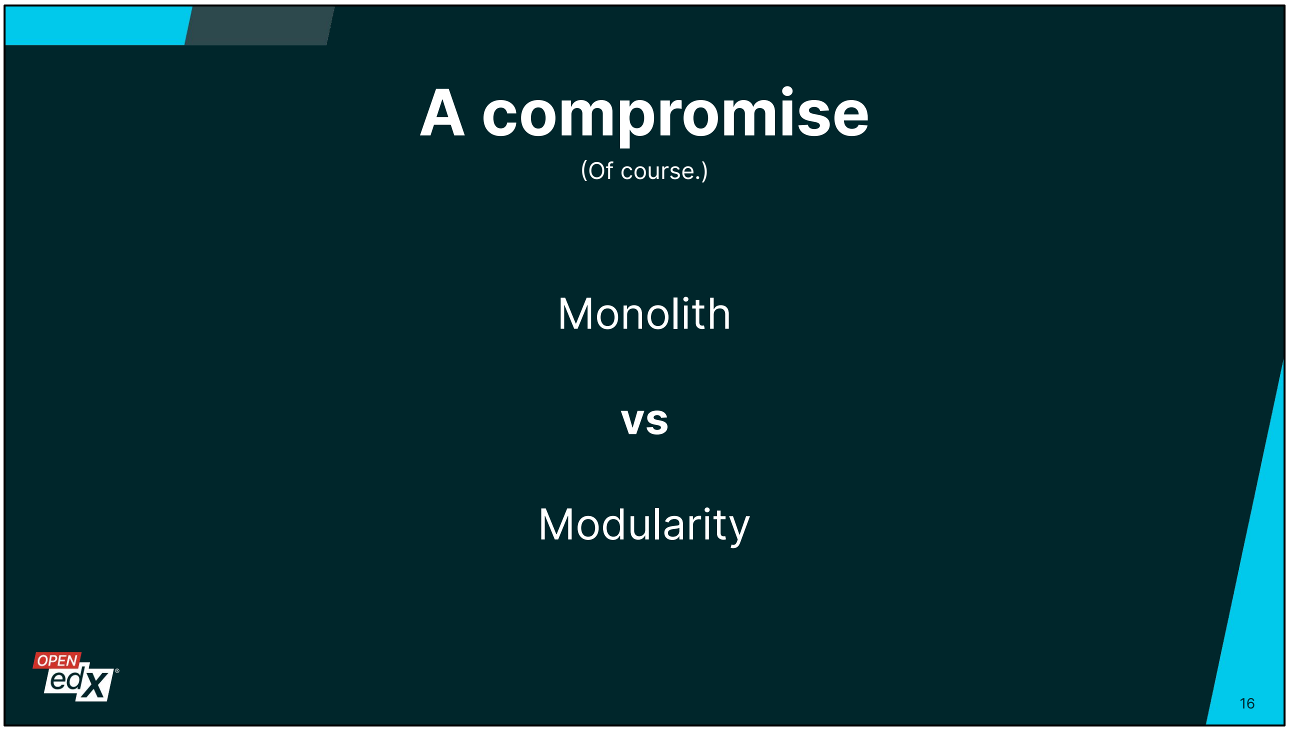
Email address (Sign in) Edit
audit@example.com

Show not tell.



Proposal: Modular Domains

Here’s what we propose to do to help things along.



A compromise

(Of course.)

Monolith

vs

Modularity

A compromise between the monolithic codebase and full MFE independence. We're going to propose sharing more between MFEs, which will help us address many of the problems listed earlier, at the cost of giving developers a little less independence. We believe a minor increase in integration costs is traded for large gains in user experience, customizability, ease of deployment, and development efficiency.

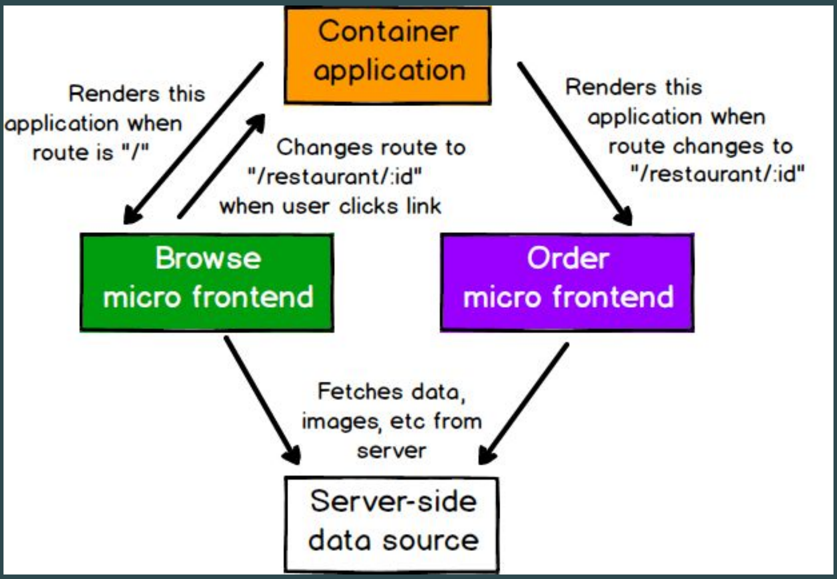
based on Domain Driven Development: the Open edX frontend will be split into formal top-level domains, including but not limited to LMS and CMS, each of which will be implemented as a Container SPA. The container, in turn, will render multiple partly-independent MFEs as sub-components. The latter will be re-engineered to conform to the hosting domain's interfaces.

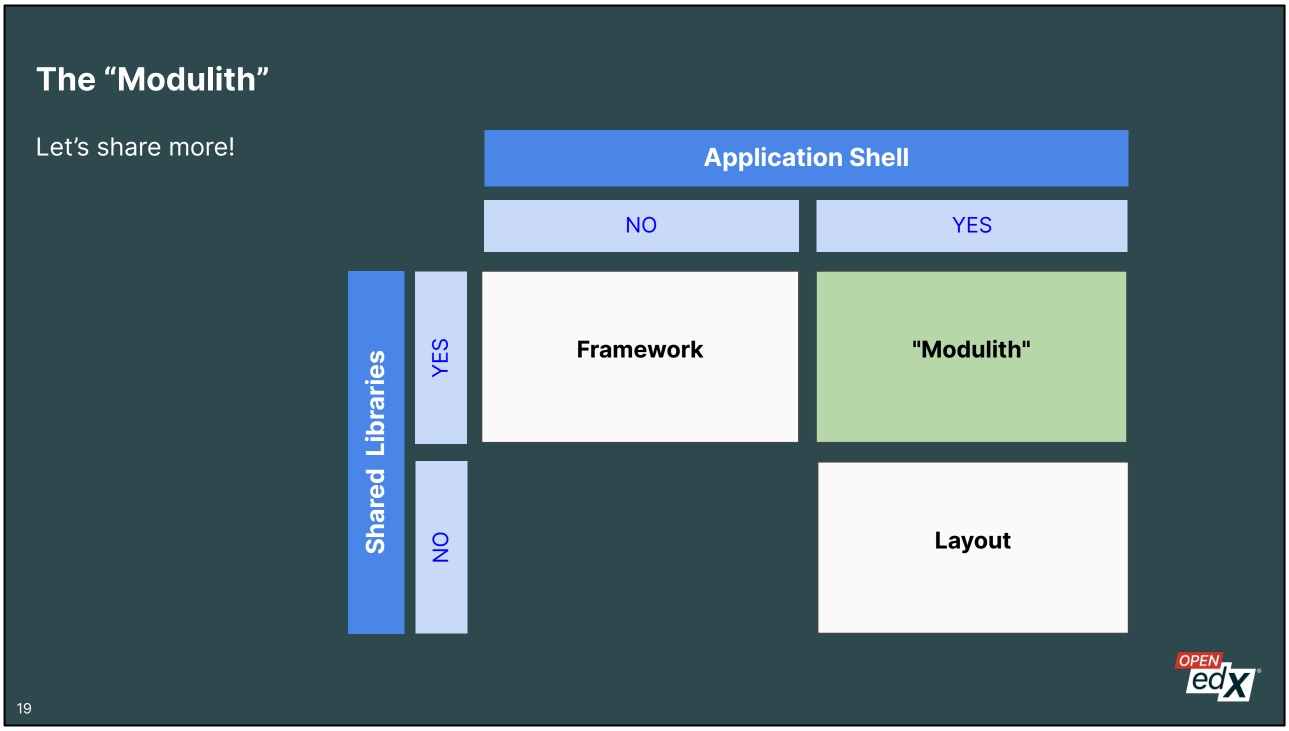
Application shells

Because sharing is caring.

Application shells

Because sharing is caring.


Container application
Renders this application when route is "/"
Changes route to "/restaurant/:id" when user clicks link
Renders this application when route changes to "/restaurant/:id"
Browse micro frontend
Order micro frontend
Fetches data, images, etc from server
Server-side data source



The "Modulith"

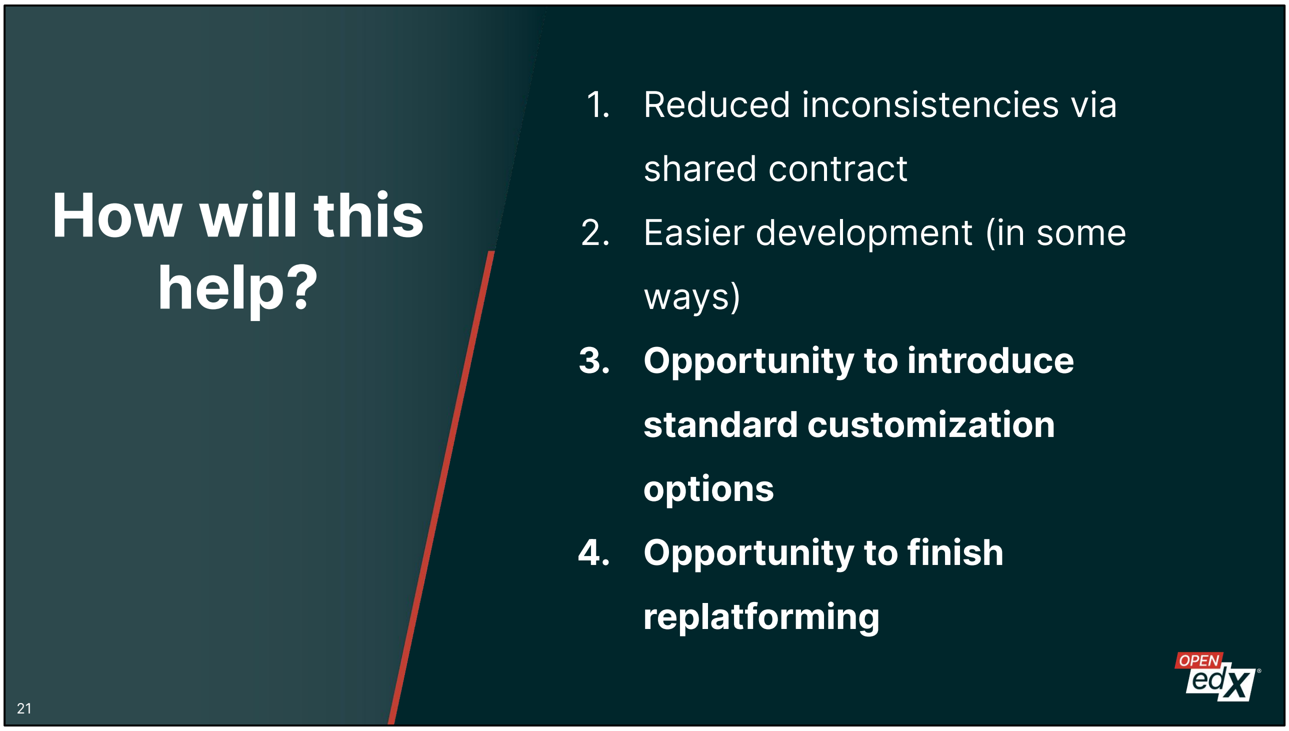
Let's share more!

We combine shared libraries and a shared design in such a way that more parts of the solution are given, while other parts can still be determined by the MFE developer.

Graphic credit: [Noah Sykes]

Application shell characteristics

- Routes between apps
- Loads libraries once
- (Hopefully) Ensures standards across apps
- **Some cross-dependency**

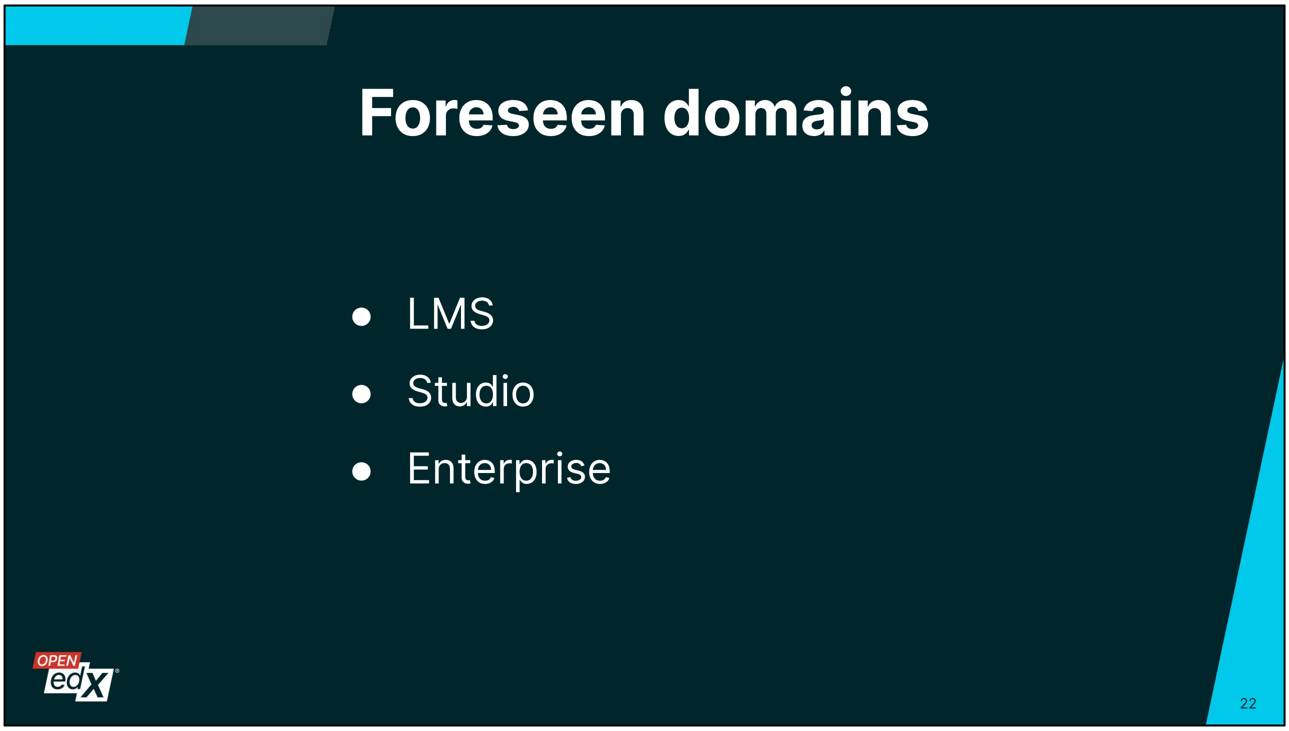


How will this help?

1. Reduced inconsistencies via shared contract
2. Easier development (in some ways)
3. **Opportunity to introduce standard customization options**
4. **Opportunity to finish replatforming**

It will be easier to develop because there’s less the developer needs to worry about. Certain elements of the page need not even be imported, such as the header. Theming and i18n will be much easier to integrate.

It will be harder in other ways: one would have to stick to a range of versions for shared apps.

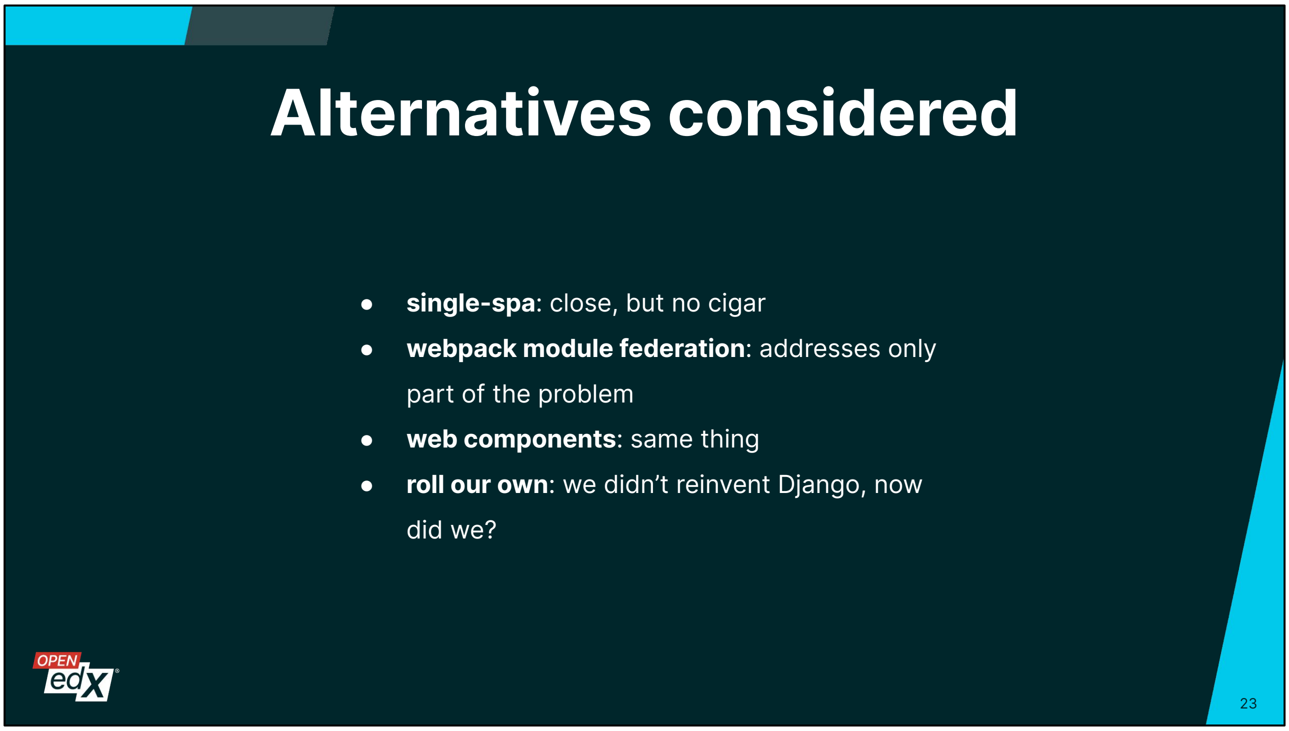


Foreseen domains

- LMS
- Studio
- Enterprise

The Open edX frontend will be split into formal top-level domains, including but not limited to LMS and CMS, each of which will be implemented as an application shell. Each container, in turn, will render multiple partly-independent MFEs as sub-components.

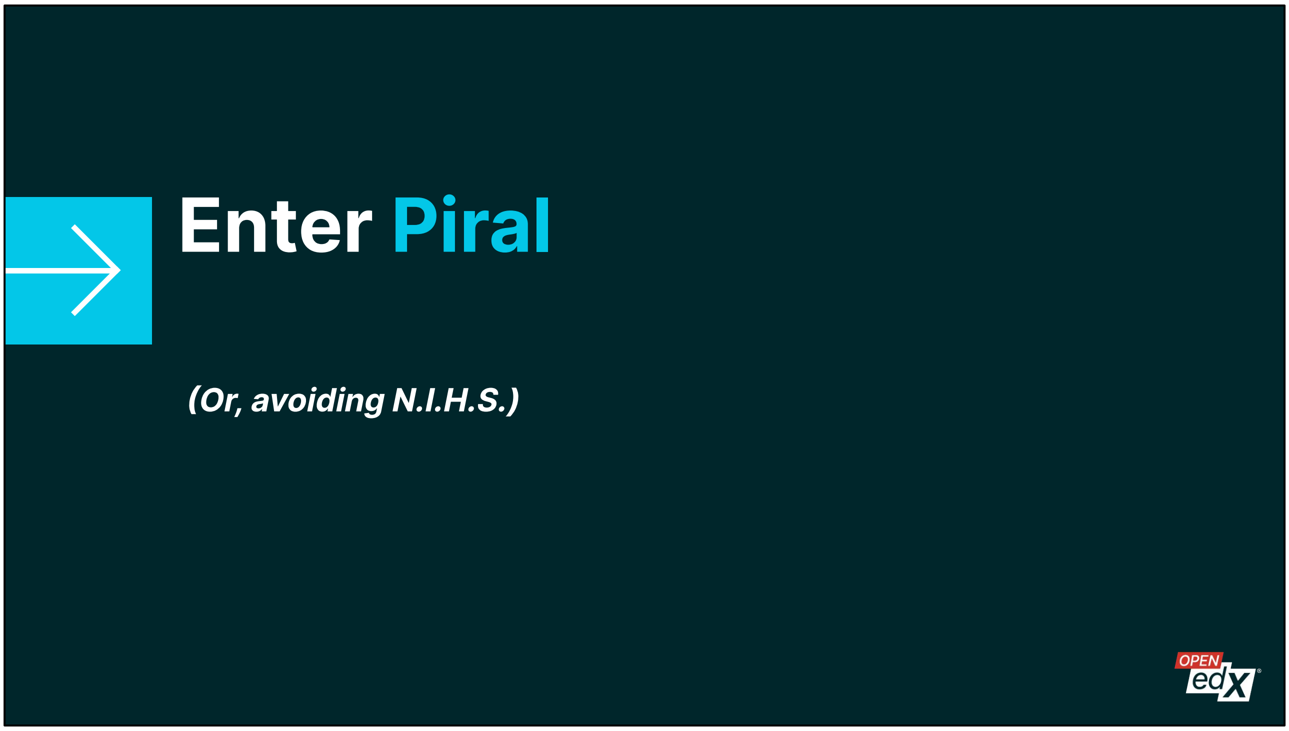
Each domain shell will share React, Paragon, and Redux libraries, for instance.



Alternatives considered

- **single-spa**: close, but no cigar
- **webpack module federation**: addresses only part of the problem
- **web components**: same thing
- **roll our own**: we didn't reinvent Django, now did we?

Single-spa was the one that came closest, but it leaves too much up to the architect. At this point, we'd rather have the community support of an active project that handles as much of the architecture as possible.

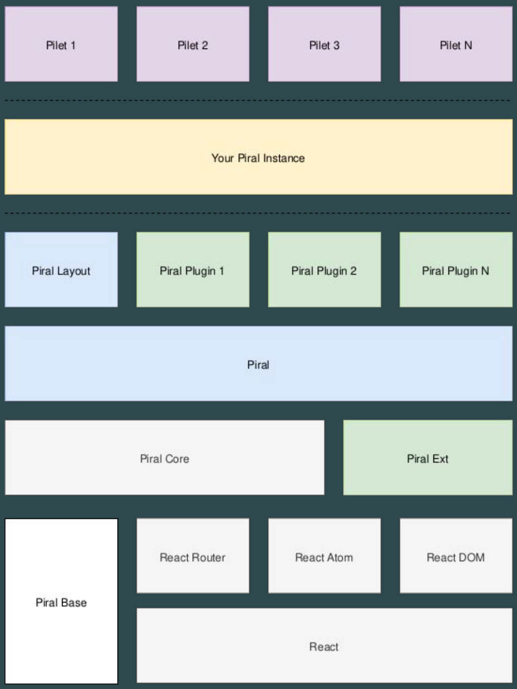


Enter Piral

(Or, avoiding N.I.H.S.)

And this is the stack we propose to use!

What does Piral do, exactly?

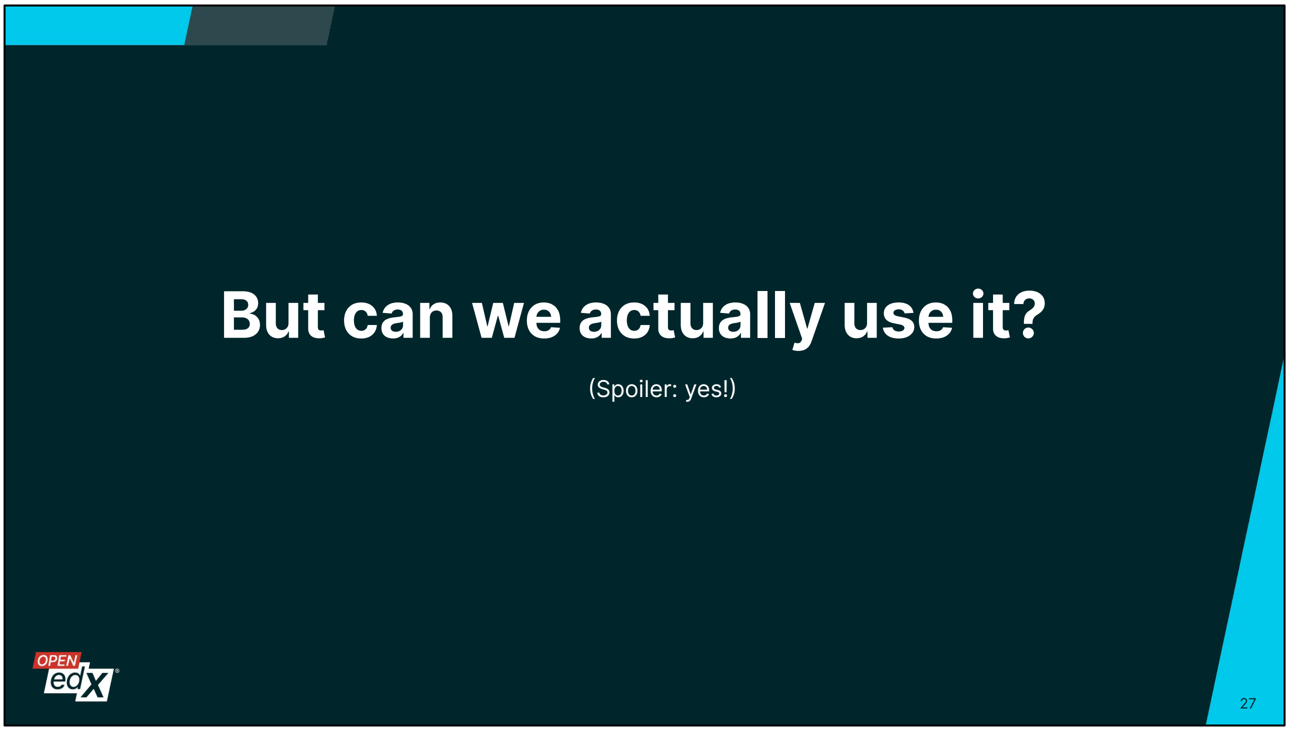


`Piral` can be thought of as a framework, while the other building blocks are just ordinary libraries. `piral-core` is like the Linux kernel. A certain distribution like Ubuntu would be `piral`. Additionally, to the kernel, there can be some special programs ("drivers"), which would be the Piral plugins. An application running in user space would then be a pilet

An opinionated solution

(Remember Django?)

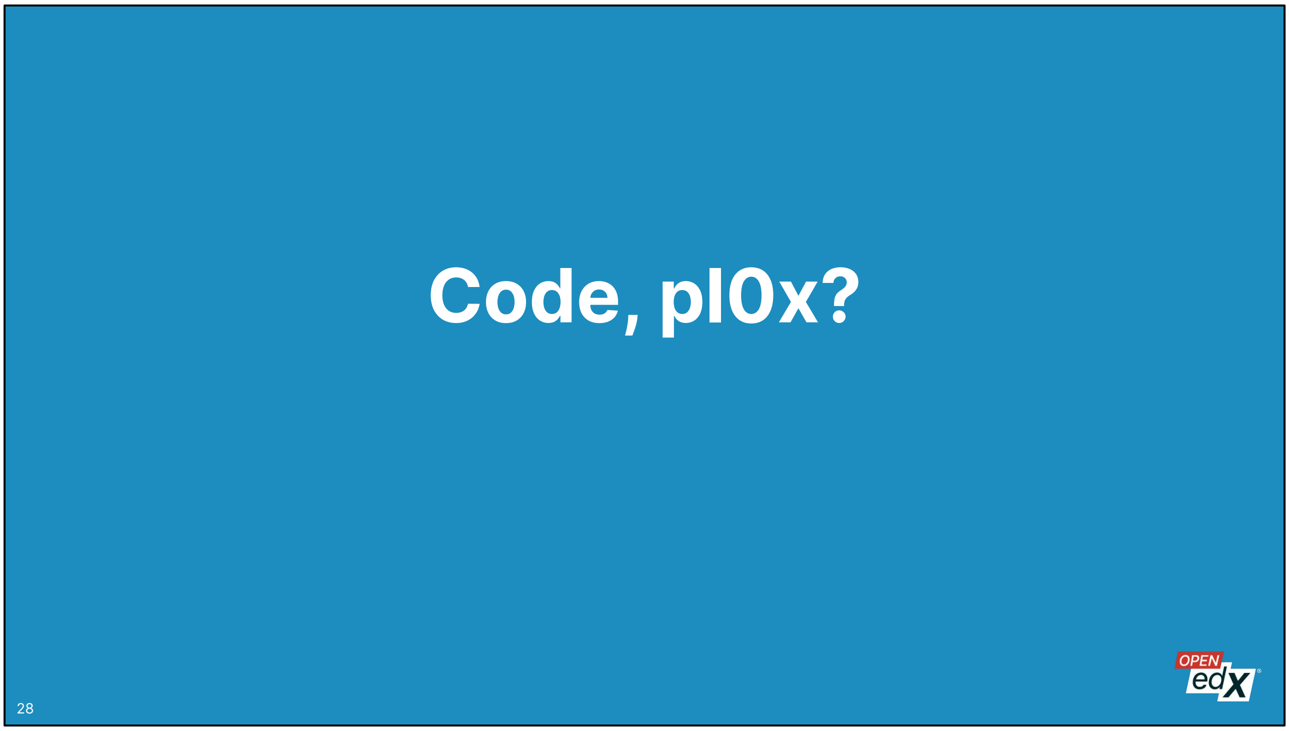
- Highly modular
- Central dependency sharing (i.e., from the app shell)
- Bundler independent (works with Webpack)
- Global state management
- Independent development and deployment



But can we actually use it?

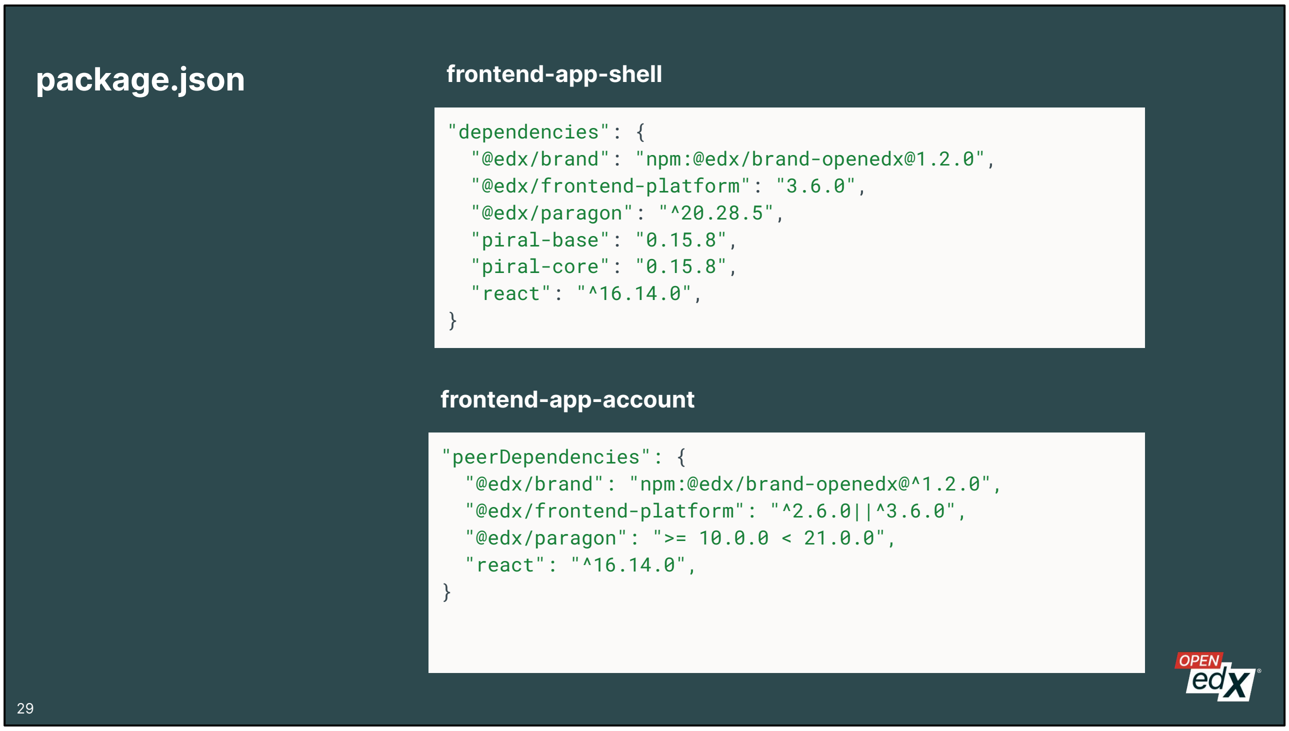
(Spoiler: yes!)

Piral is modular enough



Code, pl0x?

We’re still working on the PoC.



package.json

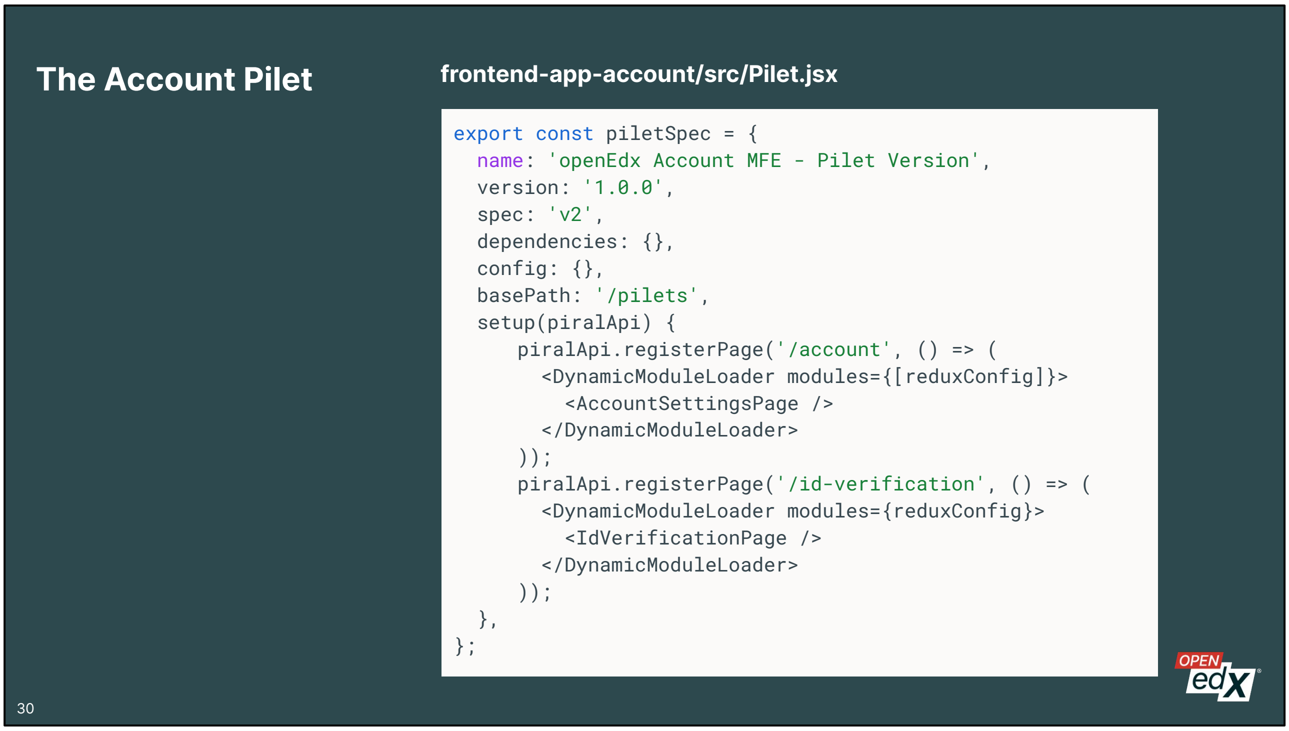
frontend-app-shell

```
"dependencies": {
  "@edx/brand": "npm:@edx/brand-openedx@1.2.0",
  "@edx/frontend-platform": "3.6.0",
  "@edx/paragon": "^20.28.5",
  "piral-base": "0.15.8",
  "piral-core": "0.15.8",
  "react": "^16.14.0",
}
```

frontend-app-account

```
"peerDependencies": {
  "@edx/brand": "npm:@edx/brand-openedx@^1.2.0",
  "@edx/frontend-platform": "^2.6.0||^3.6.0",
  "@edx/paragon": ">= 10.0.0 < 21.0.0",
  "react": "^16.14.0",
}
```

Show not tell.



The Account Pilet

frontend-app-account/src/Pilet.jsx

```
export const piletSpec = {
  name: 'openEdx Account MFE - Pilet Version',
  version: '1.0.0',
  spec: 'v2',
  dependencies: {},
  config: {},
  basePath: '/pilets',
  setup(piralApi) {
      piralApi.registerPage('/account', () => (
        <DynamicModuleLoader modules={[reduxConfig]}>
          <AccountSettingsPage />
        </DynamicModuleLoader>
      ));
      piralApi.registerPage('/id-verification', () => (
        <DynamicModuleLoader modules={reduxConfig}>
          <IdVerificationPage />
        </DynamicModuleLoader>
      ));
  },
};
```

Show not tell.

The Timeline

(Seriously, this time.)

Here’s what we propose to do to help things along.

Target

- A concerted effort by the community
- Based on the findings of the OEP
- Funded by tCRIL and other organizations
- … to migrate all Tutor-supported LMS-domain MFEs **by the end of 2023**
- (BONUS points: finish replatforming of LMS-domain edx-platform code)

OEP-XXXX

https://tinyurl.com/uavnj6mj

Questions?