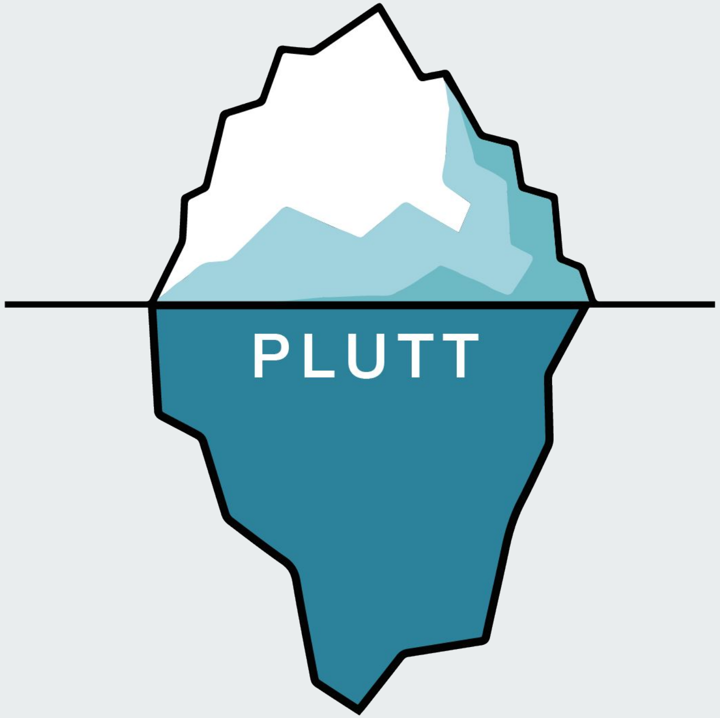
Plutt: A tool for creating type-safe and version-safe microfrontends

Julius Celik



Outline

Introduction

Introduction

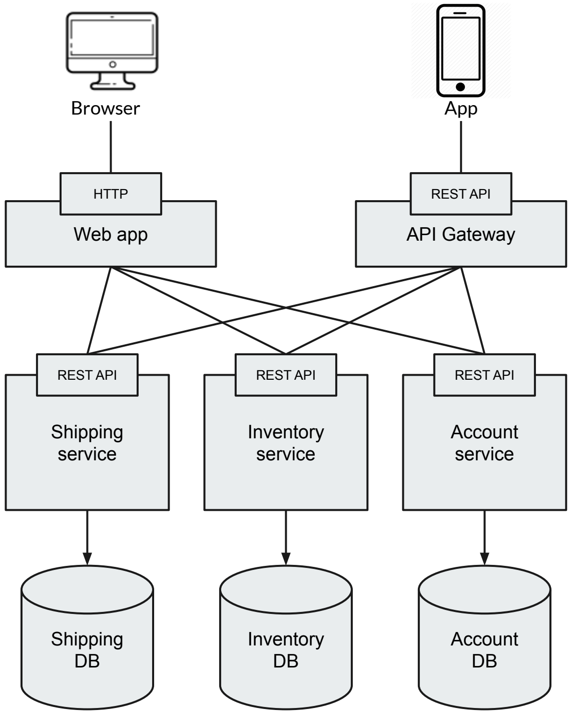
- Microservices
- Microfrontends
- Managing updates
- What I have done for my thesis

Microservices

- Widely used architecture pattern for backend systems

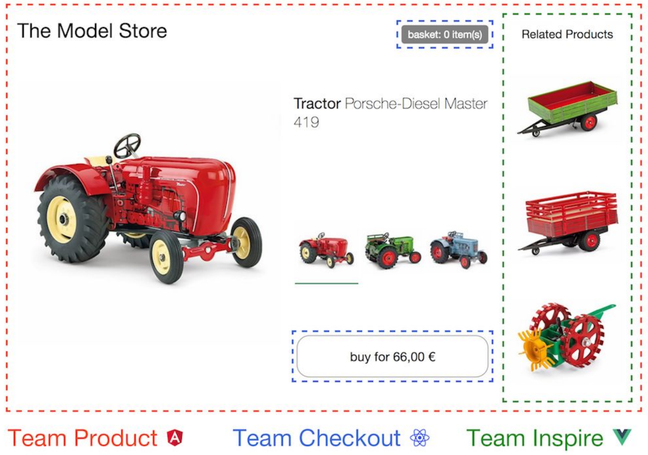
- Small **independently deployed** units
- Running on **separate processes**
- Communicating over a **network**

- The main point is providing **high team autonomy**
- This results in **low coupling**, **high cohesion**, and **strong composability**



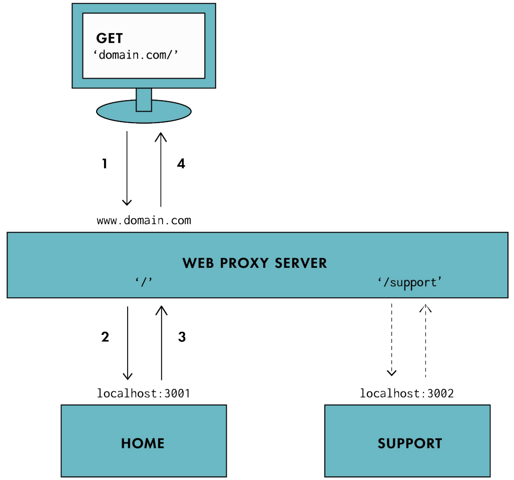
Microfrontends

- Like microservices on the frontend
- UIs consist of small frontend applications
- Microfrontends are also **independently deployable**
- Technologies can be mixed (but should be avoided)
- The **granularity** varies a lot:
  - This example is a page with many applications
  - You can have one application consisting of many pages



Different Microfrontend Solutions

- Iframes
- A web proxy (Linked SPAs)
- Web components
- Single-spa



Categorising all solutions

I distinguish solutions based on two factors:

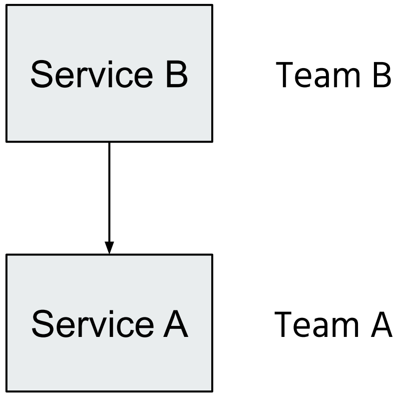
- Granularity
- Where they are composed into one coherent frontend

	PAGE	FRAGMENT
SERVER	LINKED PAGES LINKED SPAS	SERVER SIDE INCLUDES (SSI) TAILOR (BY ZALANDO) PODIUM (BY FINN)
CLIENT		WEB COMPONENTS IFRAMES SINGLE-SPA

What is independent deployability?

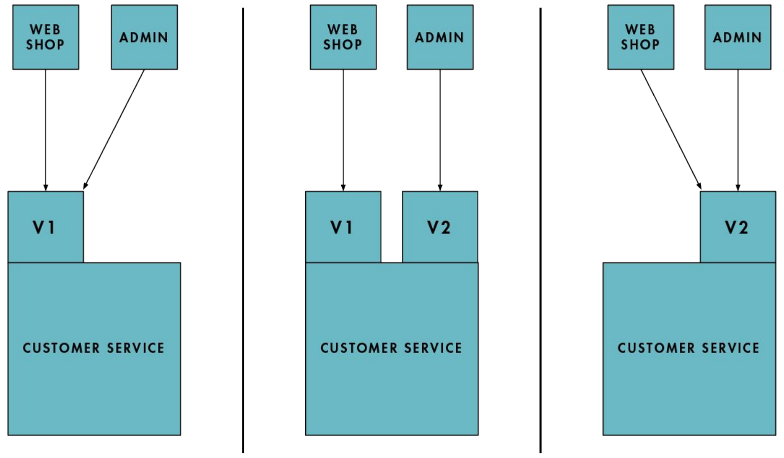
- Service B depends on Service A
- When Team A updates Service A it is deployed to production without Service B being redeployed
- Team A can deploy when it best suits them
- Team B does not have to redeploy or refactor their code every time Service A is updated

Contradicting concepts



What happens when Service A is updated?

- **Avoiding the problem:** Minimize surface boundaries by clever decomposition
- **Work around the problem:** Use **lockstep deployments** when you have to update an API



Single endpoint supported

New release exposes an additional endpoint

Once the old endpoint is no longer used, a new release of the service can remove it

Version management

- Microservices have used lockstep deployments for a long time
- Most if not all microfrontend solutions do not utilize lockstep deployments
- Version management is one of the areas that is unexplored for microfrontends

What Have I Done?

- Interview five microfrontend experts
- Develop a micro frontend tool (Plutt)
- Evaluate Plutt on three use cases

Industry Survey

Interview Protocol

- I interviewed 5 microfrontend experts
- I asked questions to understand:
    - **What** a micro frontend is: Definition
    - **Why** use them: Benefits and drawbacks
    - **How** to use them: Best practices
- I discovered aspects that can not be found in existing literature
- This can be used for future microfrontend research

Key Takeaways

- The granularity differs
- State management (don’t share state)
- **The definition is very different**
- **Performance impact of using microfrontends**

Microfrontend Definition

Three aspects, pick two:

- **Independent deployability**
- Strict **isolation** (all microfrontends are self contained)
- Organizational **alignment** to business domain

My definition:

A microfrontend is an independently deployed unit of visual UI. A microfrontend has a minimal impact on layout outside its bounding border, meaning it does not impact visual elements on multiple locations of a UI. Microfrontends are modelled to match organization structure.

Performance Impact

- A **common misconception** is that using microfrontends is detrimental for performance
- The interviewees could all share evidence of this being **false**
- Sometimes the **performance is improved** by using microfrontends

Plutt

What is Plutt?

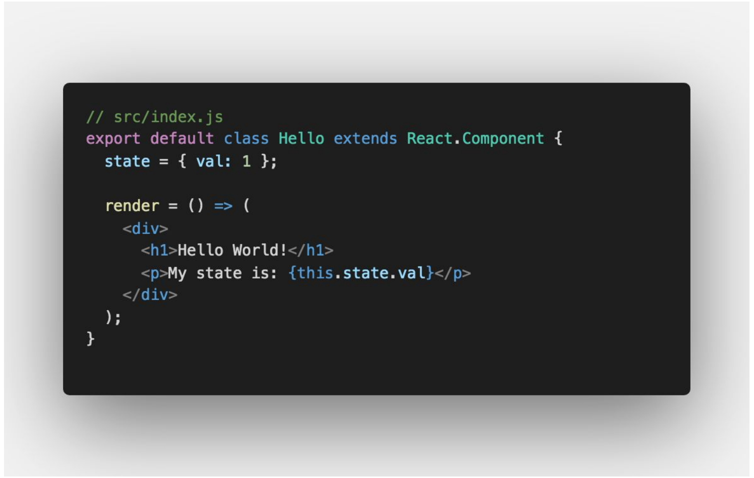
A build tool that introduces a new integration strategy concept

- Integration is done in **run-time** on the client (Like web components)
- Static information is available in **build-time** => No type errors 🤞

Plutt is:

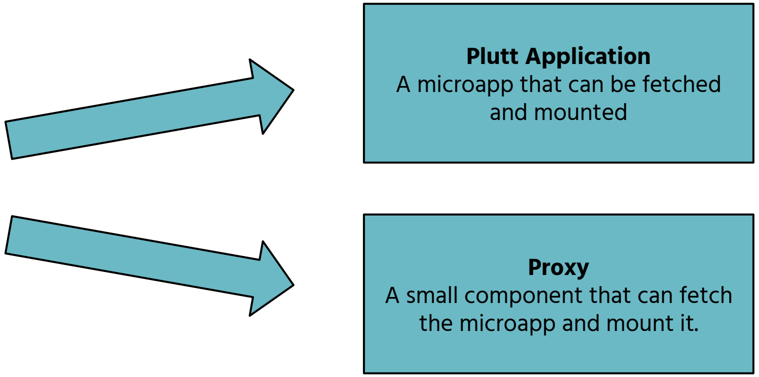
- Really **safe**: Production acts like development
- **Easy** to use: All integration logic is **generated** and **hidden** from the developer

Plutt - How does it work? (1/3)



```
// src/index.js
export default class Hello extends React.Component {
  state = { val: 1 };

  render = () => (
    <div>
      <h1>Hello World!</h1>
      <p>My state is: {this.state.val}</p>
    </div>
  );
}
```



Plutt Application
A microapp that can be fetched and mounted

Proxy
A small component that can fetch the microapp and mount it.

Plutt - How does it work? (2/3)

Plutt Application
A microapp that can be fetched and mounted

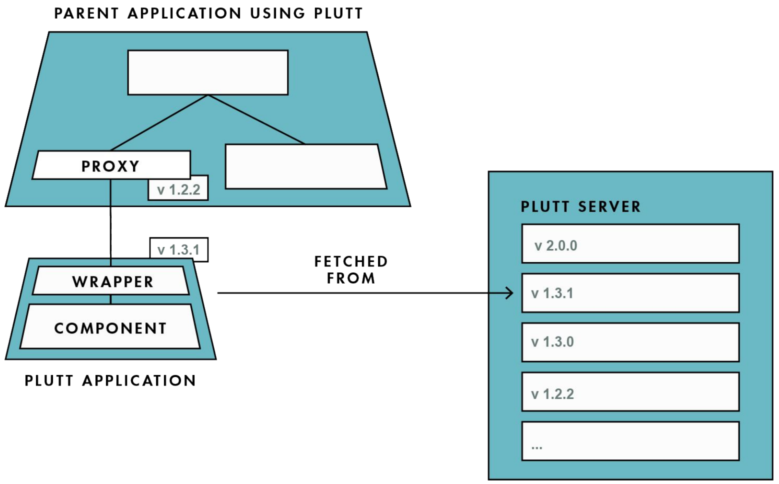
- Host this on any server
- Contains all business logic
- The micro app team can update this whenever they want

Proxy
A small component that can fetch the microapp and mount it.

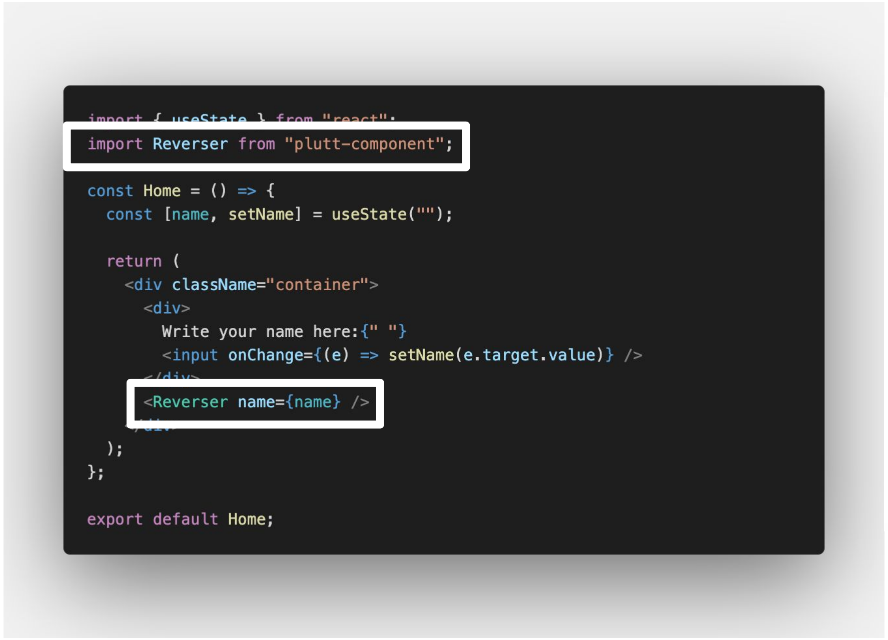
- Includes the network location of the microapp hard coded
- Contains all required logic for using (fetching and mounting) the Plutt application
- Contains no business logic
- Include this thin component in a parent at build-time

Plutt - How does it work? (3/3)

- Plutt applications can be **updated independently** from consumers
- Proxies fetches a Plutt applications, mounts it, and establishes a “communication bridge”
- Proxies acts like a **framework native** component to the parent
- A Plutt server makes sure that the proxy gets the latest update that is **non-breaking**



Plutt - Demo

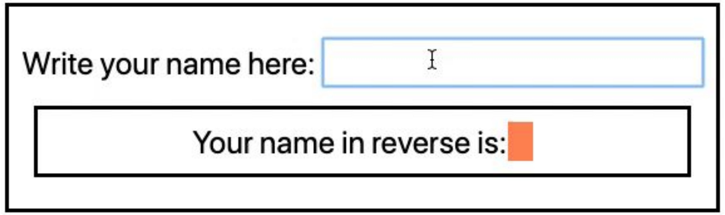


```
import { useState } from "react";
import Reverser from "plutt-component";

const Home = () => {
  const [name, setName] = useState("");

  return (
    <div className="container">
      <div>
        Write your name here:{" "}
        <input onChange={(e) => setName(e.target.value)} />
      </div>
      <Reverser name={name} />
    </div>
  );
};

export default Home;
```

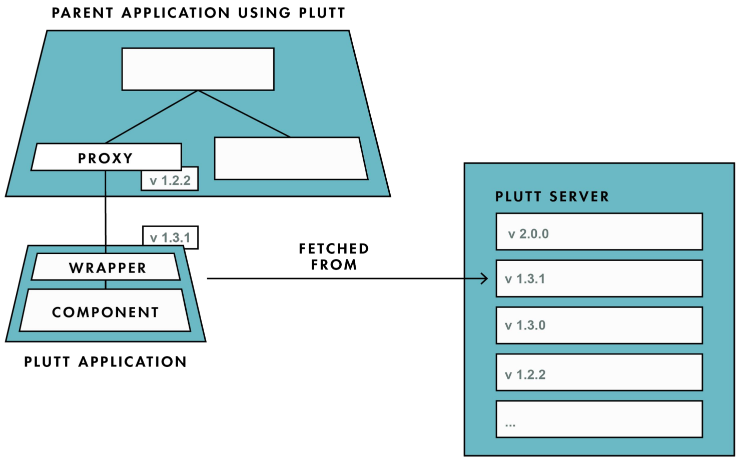


Write your name here:

Your name in reverse is:

Properties of plutt

- **Access Transparent:** Consumers don’t have to know that it is a micro frontend
- **Automatic Version Safety:** Guarantee at compile time to never use a breaking version
- **Framework agnostic consumption**
- **Type Safety:** Guarantee at compile time that you use the micro frontend correctly



Evaluating Plutt

Evaluating Plutt

I conducted three case studies:

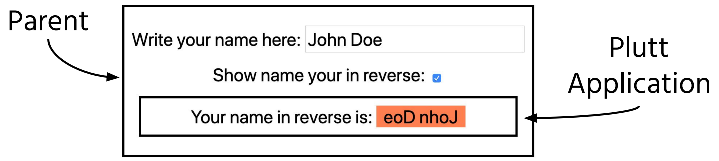
- Name Reverse: A small mock application (implementation)
- DigitalRoute: A large existing application (refactor)
- Real World Example App: A comprehensive realistic mock example (refactor)

Evaluation Dimensions

- **DX Impact:** Is it easy or hard to use Plutt
- **UX Impact:** Is the UX impacted by using Plutt
- **Correctness:** Is Plutt performing as expected

Name Reverse

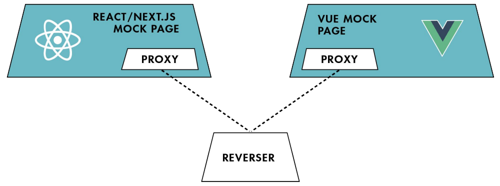
- **DX Impact:** Very easy to implement
- **UX Impact:** Feels like a monolithic application
- **Correctness:** No issues
- Implemented in both **Vue and React**, using the same Plutt Application

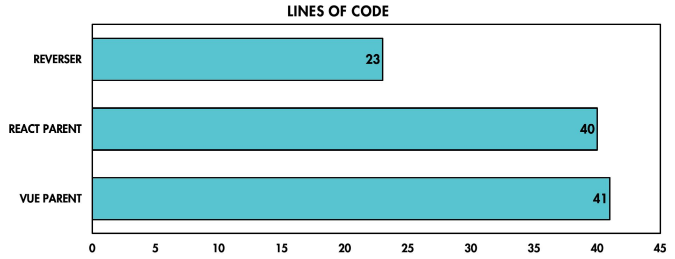


Write your name here: John Doe

Show name your in reverse: ☑

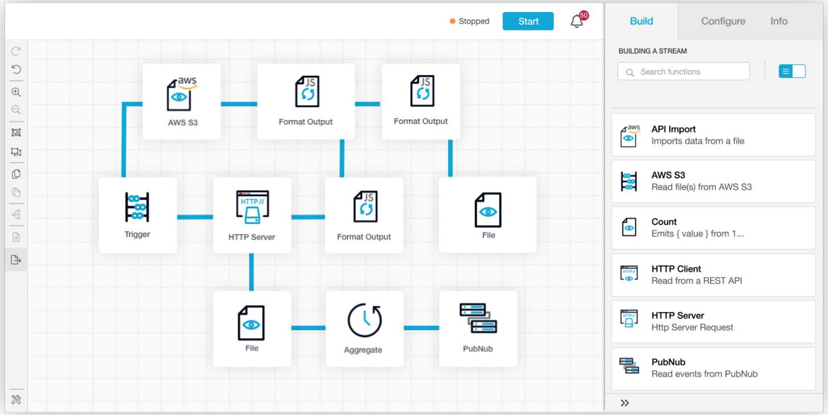
Your name in reverse is: eoD nhoJ





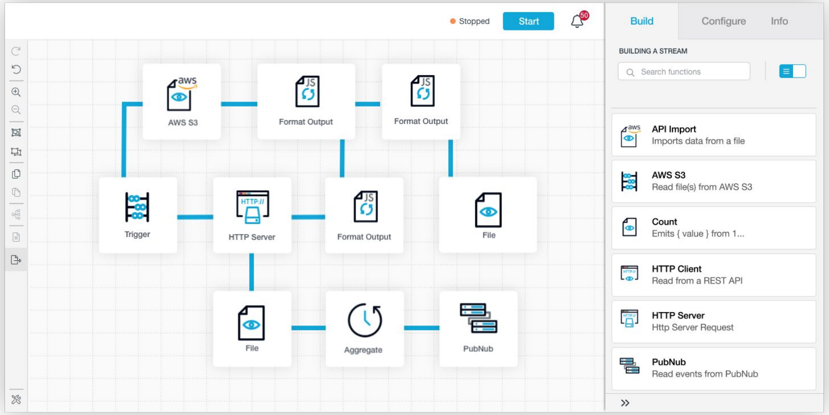
DigitalRoute (1/2)

- Heavy use of Redux, a global data store
- The data dependencies are not encapsulated into clear domains, and most components are tightly coupled
- When trying to refactor out a single form, more than half of the codebase was impacted (12 of 24 internal packages)



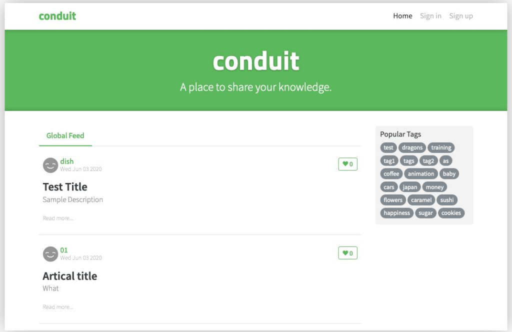
DigitalRoute (2/2)

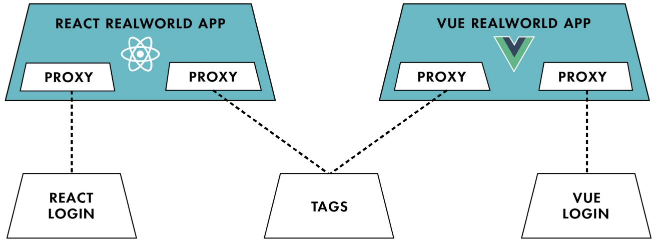
- Cohesion is low, and coupling is tight, which makes it difficult to divide the application
- The refactor could not be done
- To use Plutt there has to be major refactors to remove cross dependencies across the application



Real World Example App (1/3)

- **DX Impact:** Very easy to refactor
- **UX Impact:** No change
- **Correctness:** No issues
- Implemented in both **Vue and React**
- One **Fragment** and one **Page** is extracted

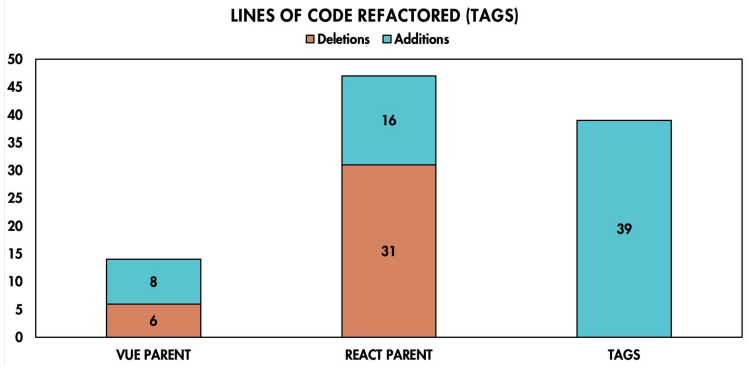




Real World Example App (2/3)

Tags:

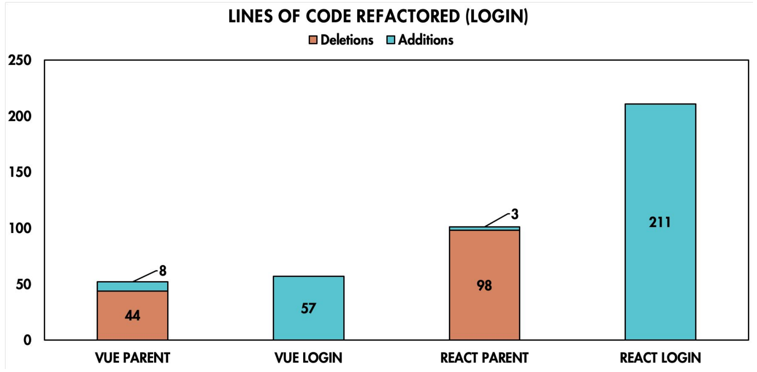
- Shared by both Parents
- Easy to refactor



Real World Example App (3/3)

Login:

- Different implementations because too large implementation disparities
- React login is a bit large, as it includes much backend API logic
- Otherwise easy to refactor



Conclusion

Conclusion

- An extensive survey of industry experts has been provided
- Plutt is a safe tool for using microfrontends
- Plutt is easy to use for many applications
- Plutt should not be used together with Singletons (like global central data stores)

Questions