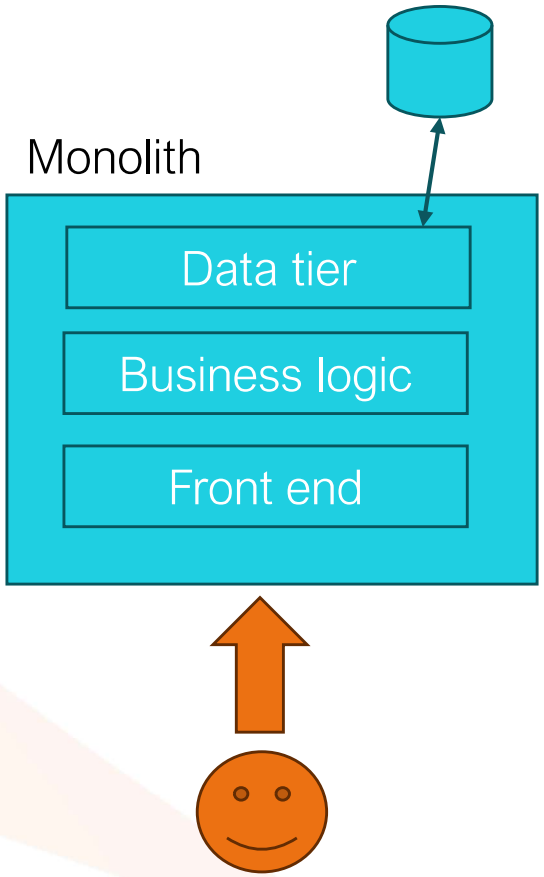
neueda

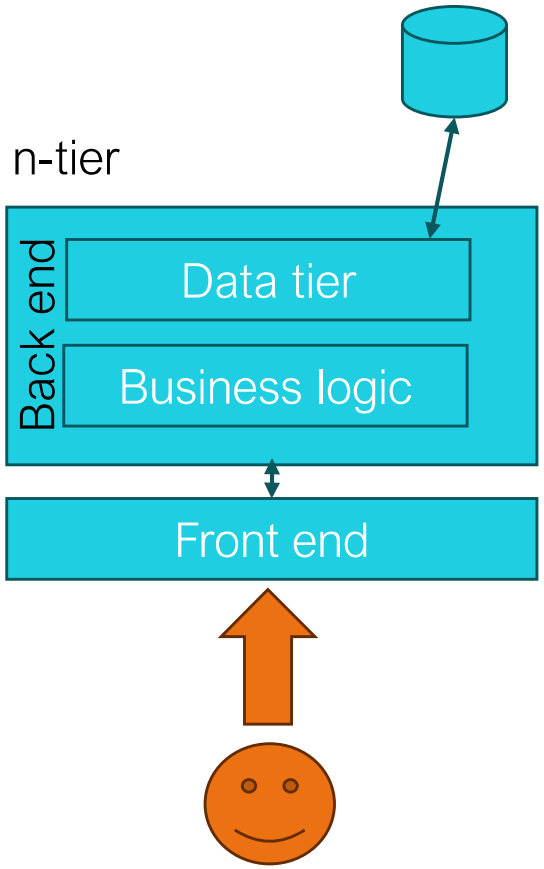
What are MFEs?

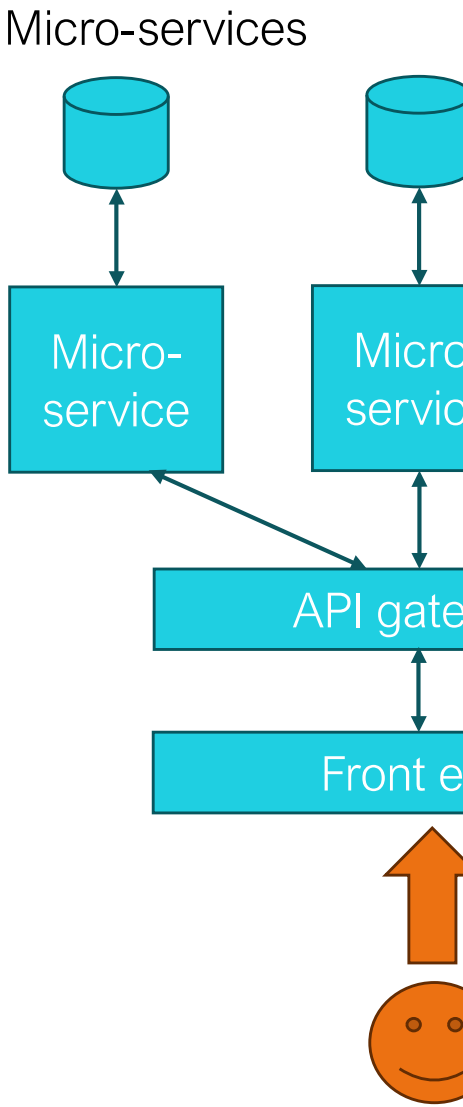
Objectives

- What are MFEs?
- The challenges of building MFEs
- MFE Architecture principles
- MFE Rules and best practice

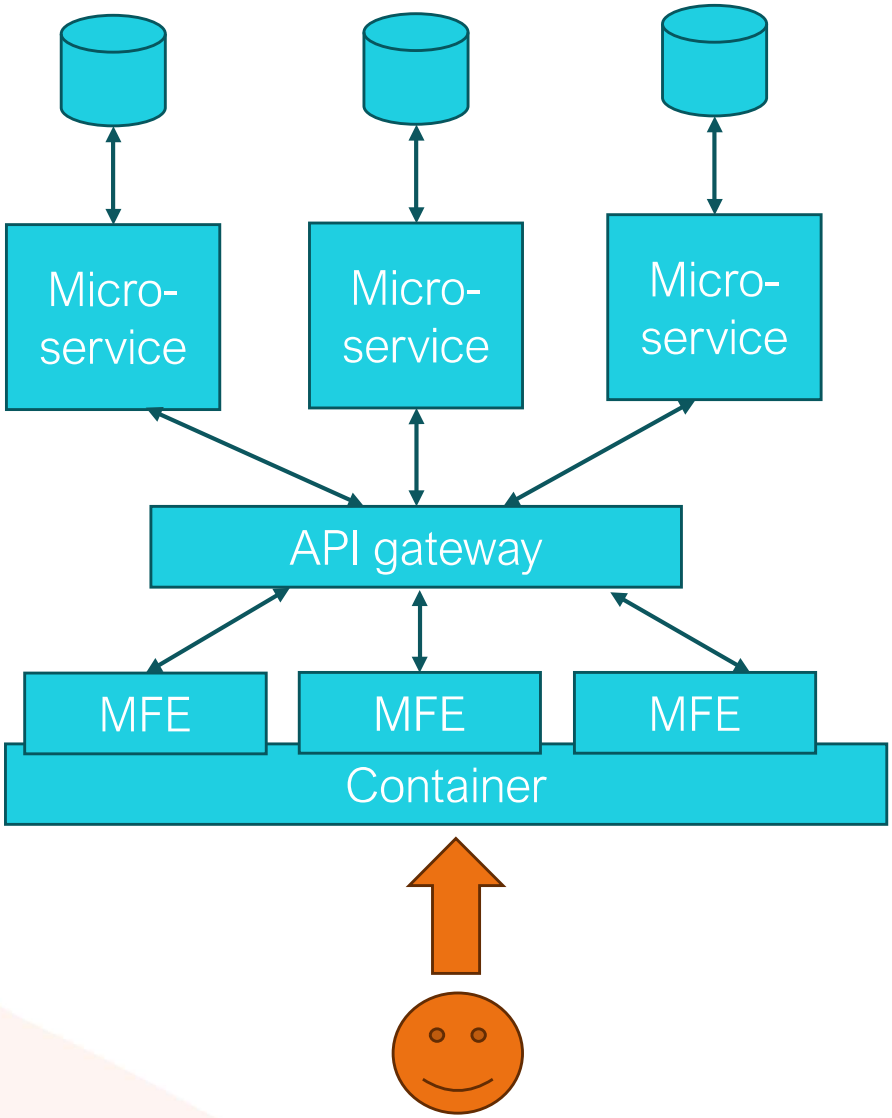
What are MFEs?







What are MFEs?



What are MFEs?

- Idea first developed in 2016
- Split a single front-end system into parts that are:
  - Self-contained
  - Loosely coupled
  - Independently deployable
    - But only if you are implementing run-time integration
- The main benefits are:
  - Easier separation of work across multiple teams
  - Quicker / less risky change processes
  - Less frustrating development processes
  - Finer-grained scaling
  - Greater resliance
  - Opportunity for progressive upgrades
  - Opportunity for mix of technologies

The challenges of MFEs

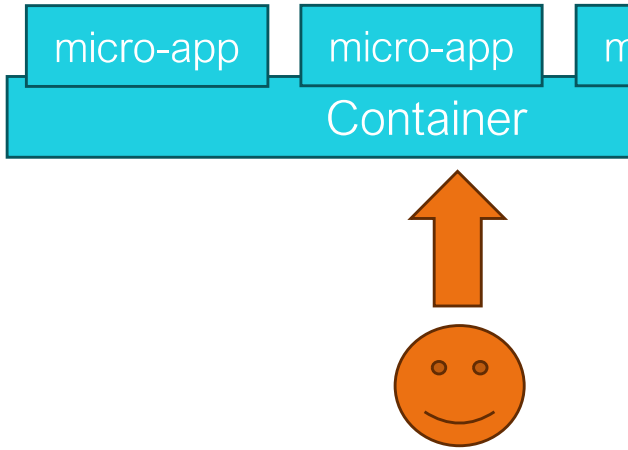
- There is no standard approach or best-practice. Each application ne
  consider and make choices around:
  - UX Consistency
    - Layout and styling
    - Interaction patterns
  - Application splitting
    - Domain driven design
    - The largest possible split that can be independently coded and deployed
    - Team splits
    - Forecast level of change
    - Scalability

The challenges of MFEs

- There is no standard approach or best-practice. Each micro-app nee consider and make choices around:
  - Micro-app Interaction
    - Stateful data in the container
    - URL embedding
    - Local storage
    - Back-end state management
  - Routing
    - URL design
    - Navigation implementation
  - Authentication
  - Framework independence

MFE Architecture principles

- Each micro-app is a separate application
  - In its own repo / in a shared repo
  - With its own build pipeline
  - Deployed to its own server / container
- The container is another application
  - It knows about the micro-apps and determines which micro-app to load and display based on its state
- Users interact with the container which dynamically loads the micro-apps as required



MFE Rules

- No micro-app can manipulate a DOM element that it did not create
- Use micro-app specific namespaces (avoid global namespace pollut
- No micro-app can rely on global state
- Interactions between micro-apps should be event driven
- Data exchanged between micro-apps should be simple
- Be conscious of performance, e.g. optimize bundle sizes

Other Considerations

- Client side rendering vs Server side rendering
- Mono-repo vs Poly-repos
- Multi-SPA vs Micro-Apps
- Single Micro-app load vs multiple micro-app loads

Building an MFE – process overview

- Create micro-apps with a local runtime (mock of the container)
- Expose the micro-app
- Create the container app, using the micro-apps
- Create the routing mechanism
- Create the global state and event driven communicaiton

Summary

- What are MFEs?
- The challenges of building MFEs
- MFE Architecture principles
- MFE Rules and best practice

neueda

Creating a basic MFE

Objectives

- Creating a micro-app
- Exposing a micro-app with module federation
- Creating a container and loading external modules

What is Module Federation?

- Module Federation is a general technique for developers to share
  between applications
- Using Module Federation with React allows us to make our compo
  available to use elsewhere
- Available for use via a Node Module called Webpack

Creating our first micro-app

The general process we will follow is:

- Create a regular React application
- Create the component(s) to be exposed
- Add the webpack dependencies into the project
- Create a config file for webpack (specifies which components are to exposed)
- Change the startup & build scripts to use webpack to build our appl
- Create code to export the component as a mountable element

Creating our first container

The general process we will follow is:

- Create a regular React application
- Create any top level component(s)
- Add in the webpack dependencies
- Create a config file for webpack (specifies which components are to imported, and where to find them)
- Use the imported components
- Change the startup & build scripts to use webpack to build our appl

Making things more standard

- Wrapping a remote component in a local component makes the co
  manageable

Practice activity 1

We are going to re-create the payments application as a micro-front-

- Create two micro-apps called “payments-list” and “payments-add” - will eventually contain the code for the two pages of our applicatio now put some dummy placeholder text in there (DO NOT PUT ANY ROUTING IN!)
- Create a container app which imports both of these as remote com Within the container app, you can create a menu and use routing to determine which of our two remote components to display. (re-use code from the payments-ui application as appropriate).
- OPTIONAL – create a further micro-app to serve static pages such a home page and the 404 page

Summary

- Creating a micro-app
- Exposing a micro-app with module federation
- Creating a container and loading external modules

neueda

Dependency Managemer

Objectives

- Avoiding duplication of module loading
- Module version management
- Single version loading
- Delegating module selection

Avoiding duplication of module loading

- We can mark modules as shared dependencies
- The container will inspect each of the remoteEntry.js files for all c apps and will decide which copy to load
- The single copy of the loaded module is then made available to al apps

Module version management

- Sometimes micro-apps will use different versions of modules
- Webpack will automatically only share modules where the required
  are identical

Single version loading

- Even if we share versions of modules, they will still be loaded into m multiple times.
- Where you have particularly large modules this can lead to perform degradation
- You can specify that a shared module should be loaded once in mem declaring it as a singleton

```
shared: ['react', 'react-dom', 'css-loader', 'ts-loader'
{ myModule : {singleton : true}}]
```

Delegating module selection

- Webpack can be configured to automatically share all dependencie container application
- This means that you do not need to explicitly mark them as shared remote applications
- Although this is less work, it removes fine-grained control

The steps are:

- Split the dependencies in package.json into dependencies and devDependencies
- Import package.json into the webpack.config.js file
- Use this to determine the shared modules

Summary

- Avoiding duplication of module loading
- Module version management
- Single version loading
- Delegating module selection

neueda

Inter-app communicatio

Objectives

- The event driven communication model
- Using functions for state-change
- Activity 2 – practice communication
- Updating state from a parent

The event driven communication model

- Communicating between the container and a micro-app is similar communication between a parent and child component:
  - The micro-app can accept properties
  - The properties are a js object containing data and functions
  - We can use the functions to pass data to the parent (container)

- Direct communication between micro-apps is not possible (or recor

Using functions for state-change

- You cannot pass a change of state from the container to a micro-ap initial value of its properties
- If we want to pass a change of state in either direction we need to u functions.
- YOU SHOULD MINIMISE THE DATA PASSED

Activity 2 – practice communication

- Create a context in the container to store the state of the current u
  whether they are logged in or not.
- Use the state of the user to determine whether or not to show the
  (pass the state to the relevant component)
- In the home page, create a login / logout button. The button should
  container that the user has logged in / logged out
- Do not try and update the state of the child component

Updating state from a parent

There are various possible strategies for passing details of a state ch
a parent to a child:

(1) Provide a function to the child that the child can call to obtain the
state and poll that function as required

(2) Provide a notification of some change in the route parameters

(3) Create a shared context

(4) Use local storage / indexeddb

(5) Providing a callback function

Updating state from a parent

Container

```
const mountReturn = mount(el, props);
```

Remote

```
type mountReturn =
{ someFunction : () => void }

const mount : mountReturn = ()
```

Summary

- The event driven communication model
- Using functions for state-change
- Activity 2 – practice communication
- Updating state from a parent

neueda

Routing

Objectives

- Why routing is not quite so straight forward
- The MemoryRouter component
- Returning a callback function

Why routing is not so straight forward

- React will not let us implement a BrowserRouter inside a Browser
- When navigating in a mfe, we don’t know the URL of the containe
- When an mfe navigates, the container doesn’t know automatically

The MemoryRouter component

- A memoryRouter is a standard react component which stores its lo memory rather than the browser history
- We can use a regular BrowserRouter in the component and a Memo in all the mfes that need routing.
- We then need to code up:
  - When a routing change happens at the container level, let the MFE kn
  - When a routing change happens at the MFE level, let the container kn
- These changes are achieved through functions

Returning a callback function

- When a container needs to tell a remote component that the URL h
  changed, the remote component needs to return a suitable callbac
  from the mount function (like in the previous chapter)

Activity 3 – implement routing

Warning : This is a tricky challenge!

- Implement the “Find a transaction” feature by copying the code fro
  previous react project

Hint – the initial path to pass from the container when mounting the

```
initialEntries: [location.pathname+location.search]
```

Summary

- Why routing is not quite so straight forward
- The MemoryRouter component
- Returning a callback function

neueda

Other Considerations

Objectives

- Lazy Loading
- CSS
- Authentication

Lazy loading

- Rather than loading all the javascript when the container first loa
  make some of the components lazily loaded so that they are load
  needed.
- This will potentially provide better performance, especially in larg
  projects.
- Note that this is a standard React feature!

CSS

- In React, css is globally scoped (not specific to a component)
- In a MFE, as css rules are loaded, they are applied globally
- You must therefore be careful to use naming conventions for css ru
  avoid collisions / overrides

Authentication

General best practice approach is:

- Store the user state in the container
- Create a MFE specifically to handle the login / logout process
- Share the user’s state from the container to all MFEs, including any tokens

Summary

- Lazy Loading
- CSS
- Authentication