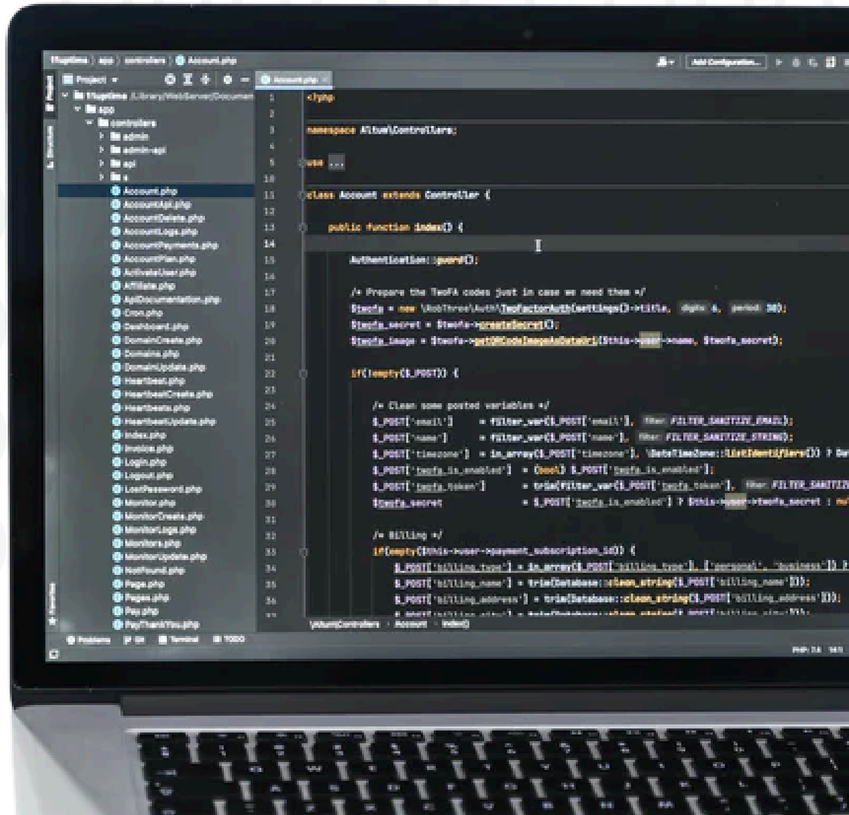
MICRO-FRONTENDS WITH SINGLE SPA

Elite
IT Academy



TOPICS

- **Introduction**

Elite
IT Academy

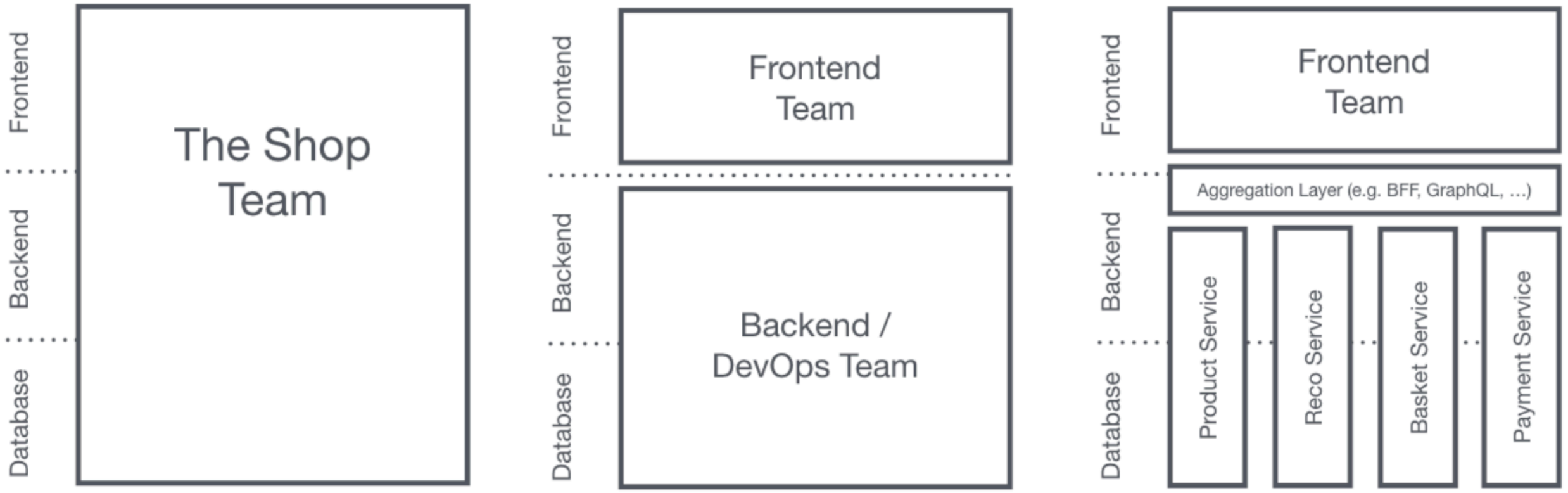
Introduction

Elite
IT Academy

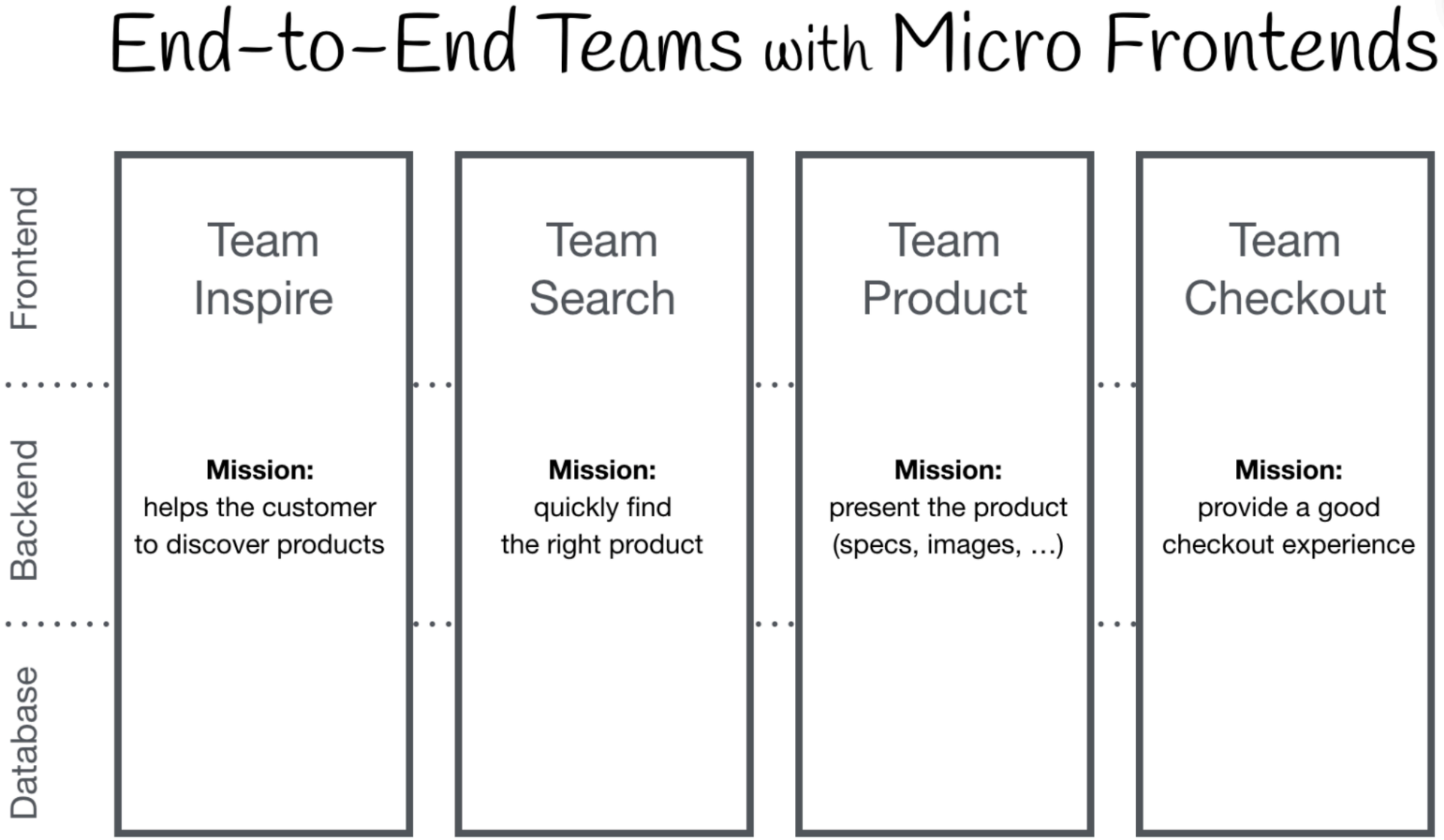
Introduction

- Micro Frontend (MFe) is a web application implementation design idea that breaks the traditional monolithic way of implementing large or midscale web applications.
- MFe allows teams to work separately with their frontend technology and addresses a common problem.
- The idea behind Micro Frontends is to think about a website or web app as a composition of features which are owned by independent teams.
- Each team has a distinct area of business or mission it cares about and specialises in. A team is cross functional and develops its features end-to-end, from database to user interface.

Introduction



Introduction



Root Application

- The root application needs to be shared by every MFe application.
- It has an HTML file that is a placeholder for all your MFe apps.
- The root application registers all the MFe apps and is responsible for bootstrapping, mounting, or unmounting the MFe into the DOM.
    - ***import { registerApplication, start } from 'single-spa';***
- The register application function is used to register all your MFe apps.
- The start function will not be bootstrapped, mounted, or unmounted until the application is loaded.
- The entire process is known as the configuration setting.
- There are many different functions that provide more control over the flow of the design, such as the triggerAppChange, navigateToUrl, getMountedApps, getAppNames, getAppStatus, unloadApplication, unregisterApplication, addErrorHandler, removeErrorHandler, pathToActiveWhen etc.
- Note that the root app configuration is the most important step of the MFe design

Micro-Frontend Types

- MFe apps should be handled with routes as much as possible.
- The MFe apps don’t need to share the UI state.
- The applications will always be created and destroyed when they share the same route. Therefore, you shouldn’t share UI states between MFe apps.
- The single-SPA has categorized the MFe design into three parts, including:
    - **Application:** The application is the core building block of the MFe design. When it’s integrated with the root application, it will serve a specific purpose.
        - For example, the application may export public interfaces like components, UI state, value, parcel, or methods – and it must have its routes.
        - It should also manage its lifecycle when it gets mounted, unmounted, and bootstrapped.
        - These may also have interfaces that have to be imported by an MFe application.

Micro-Frontend Types

- **Parcel:** The parcel is like a web component.
  - It does not have any routes and requires a custom lifecycle, which means the other application will have control when it gets to mount, unmount, and bootstrap.
  - The parcel is best suited when a UI feature needs to be shared across applications.
  - The parcel can be small like a function or as big as an application.
  - Parcel-based MFe implementation is not recommended as it will require UI states to be shared between MFes, but it cannot be completely avoided.
  - It is useful if the MFe applications are developing on different platforms.
  - The parcel may have its own deployment strategy with its own repository, or it can be a public interface of an application.
- **Utility:** The utility applications have public interfaces.
  - It doesn’t have any routes or UI to be rendered. Instead, it contains the logic that must be shared across the MFe applications.

Best Practices

- The MFe design should be route-based to reduce the maintenance of the UI states across MFe apps.
- Create the parcel if the same UI is used in multiple MFe apps.
- Create the features of data fetching, style guide, error logging, notification, etc., as a utility.
- The MFe apps may have some features shared as a public interface.
- The MFe apps should not interact or have significantly less interaction. If its frequency is too much better, merge those MFe apps into one.
- Import-map-overrides is the tool for local development as well as for testing.
- There should not be a common store (state management) across all MFe. The store can be at one application level only.
- It is better to have standard frontend technologies for all the MFe.
- Do not use mono repos

Creating a Multi-Frontend Application

Elite
IT Academy

Single spa Applications

- A single-spa root config, which renders the HTML page and the JavaScript that registers applications. Each application is registered with three things:
  - A name
  - A function to load the application's code
  - A function that determines when the application is active/inactive
- Applications that can be thought of as single-page applications packaged up into modules. Each application must know how to bootstrap, mount, and unmount itself from the DOM.
- The main difference between traditional SPA and single-spa applications is that they must be able to coexist with other applications as they do not each have their own HTML page.
- For example, your React or Angular SPAs are applications. When active, they can listen to url routing events and put content on the DOM.
- When inactive, they do not listen to url routing events and are totally removed from the DOM.

Create root config

- Invoke create-single-spa to generate a root-config by running:
  - **npx create-single-spa --moduleType root-config**
- Follow the remaining prompts with a few things in mind:
  - single-spa Layout Engine is optional at this time but is recommended if you foresee utilizing server side rendering
  - the orgName should be the same across all of your applications as it is used as a namespace to enable in-browser module resolution
- Once created, navigate into the newly created root-config folder
- Run the start script using your preferred package manager

Create root config

- Navigate to A single-spa root config, which renders the HTML page and the JavaScript that registers applications. Each application is registered with three things:
  - A name
  - A function to load the application's code
  - A function that determines when the application is active/inactive
  - Applications that can be thought of as single-page applications packaged up into modules. Each application must know how to bootstrap, mount, and unmount itself from the DOM.
- The main difference between traditional SPA and single-spa applications is that they must be able to coexist with other applications as they do not each have their own HTML page.
- For example, your React or Angular SPAs are applications. When active, they can listen to url routing events and put content on the DOM.
- When inactive, they do not listen to url routing events and are totally removed from the DOM. in your browser
- You now have a working root-config!

Create Single SPA application

- Invoke create-single-spa to generate a single-spa application by running:
  - **npx create-single-spa --moduleType app-parcel**
- Follow the remaining prompts to generate a single-spa application using your framework of choice
- Once created, navigate into the newly created application folder
- Run the start script using your preferred package manager

Add Shared Dependencies

- Shared dependencies are used to improve performance by sharing a module in the browser through import maps declared in the root-config.
- For example, if using React the generated Webpack config already expects React and ReactDOM to be shared dependencies, so you must add these to the import map. Vue, Angular, and Svelte don't require shared dependencies at this time.
  - "react": "https://cdn.jsdelivr.net/npm/react@17.0.2/umd/react.production.min.js",
  - "react-dom": "https://cdn.jsdelivr.net/npm/react-dom@17.0.2/umd/react-dom.production.min.js"
- As your architecture matures, you may add more shared dependencies in the future so don't stress about leveraging these perfectly at first.

Register the Application

- Return to the root-config and add your application to the import map in src/index.ejs
- The application's package.json name field is recommended
- Register as a single-spa application
- *if not using single-spa Layout Engine*
  - Open src/root-config.js
  - Remove the code for registering @single-spa/welcome as an application
  - Uncomment the sample registerApplication code and update it with the module name of your application
- *if using single-spa Layout Engine*
  - Remove the existing <application name="@single-spa/welcome"></application> element
  - Add your own <application name=""></application> element using the name the module name used in the import map from the previous step
- Thats it! Your first single-spa application should now be running in your root-config.