WHITE PAPER

MICRO FRONTENDS

Abstract

Over years, quick turnaround on application development/ enhancements through reduced efforts that aim at highly maintainable applications have become utmost necessity over result being development of just fully functional applications. Industry has come a long way from large backend systems to granular micro-services to try to progress in this direction. Further, advent of Micro-services has resulted into evolution of monolithic frontends to micro frontends to take it to the next level.

This paper presents point of view on what Micro Frontends are, how it is evolved, its principles and where Micro Fronts are more relevant. It provides various approaches to implement Micro Frontends, trends, and its Maturity model.

Infosys®
Navigate your next

Table of Contents

Introduction **3**

Evolution of Frontends **3**

Why Monolithic frontends aren't advisable? **4**

Principles of Micro services and hence Micro frontends **4**

Where to leverage Micro frontends **5**

Approaches to implement Micro Frontends **6**

Approach for isolated micro-apps: 6

Approach for micro-apps on common runtime: 6

Comparison of Micro Frontend Approaches **7**

Maturity Model of Micro frontends **8**

Summary **8**

About the authors **8**

Appendix: Deep Dive into Web Component based Micro Frontend Approach **9**

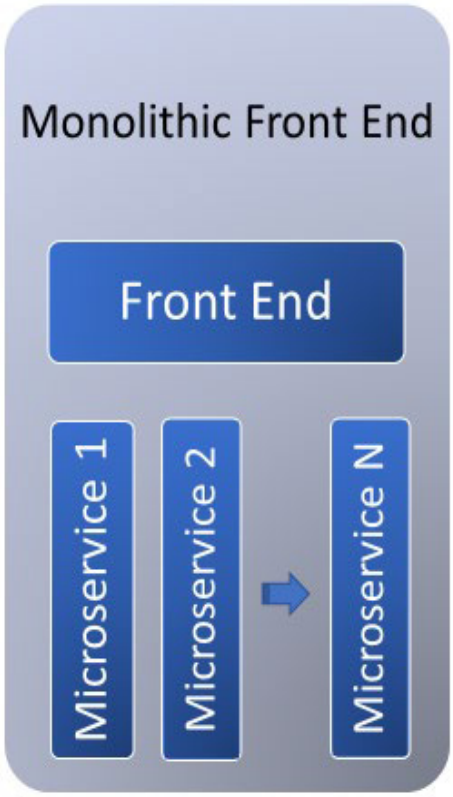
References **15**

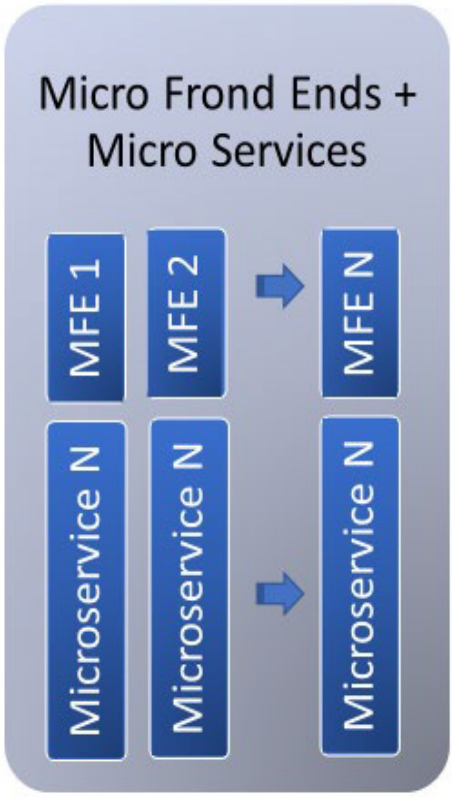
Introduction

Micro frontend is forthcoming architecture style for user interfaces. It was evolved from microservices architecture of the application backend that helped apply its benefits to front end applications. Collection of individual Micro frontend applications together compose a greater application, and it enables delivery of specific micro frontend application at a time. Approaches to implement Micro frontend can be as Isolated Micro Apps or on Common Runtime. A micro frontend can be composed of horizontal or vertical split. Micro frontend is an evolving architectural style and hence more open-source JS based frameworks are coming up in the market. A choice of framework for implementation purely depends on organizational standards when it comes to adoption of a specific approach/framework.

Evolution of Frontends

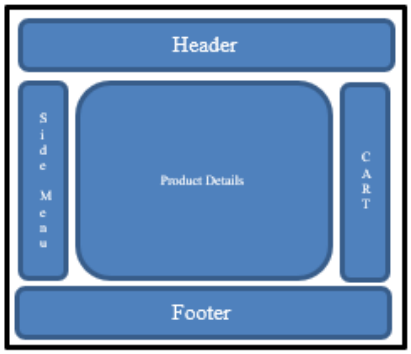
Typically, application is divided into frontend, backend, and data store layers. Decades ago, front end and backend components were combined into one single monolithic application where a change in a user interface used to have cascading effect on its business logic that was residing in backend. Slowly frontend and backend portions of an application were divided to achieve better skill alignment, change management and application maintainability. This then evolved into one backend system being leveraged by multiple front-end systems to achieve reusability but that contributed to scaling issues for backend systems. Hence microservices based architecture was developed to address scaling issues with the help of large backend systems split into smaller domain driven micro services and it resulted into front-end application calling multiple smaller backends. These nuclear micro services caused realization of the fact that frontend applications are still monolith and hence a necessity to solve issues in frontends with the help of microservices design principles, this is when Micro frontend architecture came up.



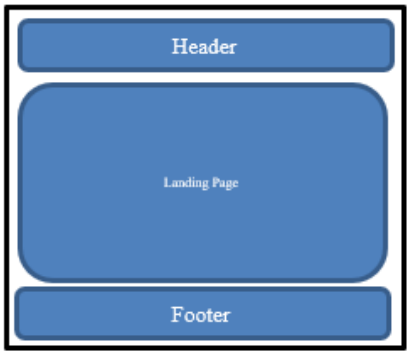


Logical approaches of structuring Micro Frontends are horizontal or vertical split. Horizontal split is when one or more micro frontends are presented within the same webpage side by side whereas vertical split doesn't present more than one micro application in a webpage at the same time. Following diagrams depict how horizontal split shows multiple micro apps rendered through a base/ shell app and how vertical split shows only one micro app within a base/ shell application.

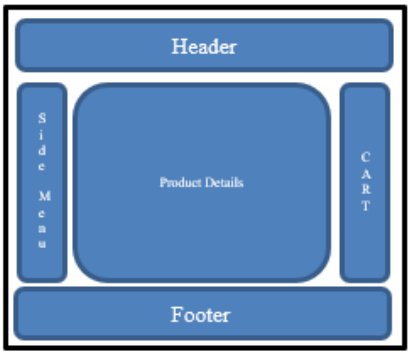
Horizontal Split



Vertical Split



Micro App1



Micro App2

Why Monolithic frontends aren't advisable?

Monolithic frontend applications aren't preferred as:

- They have huge codebase built over the period hence, it becomes less maintainable. This causes small change to take longer to complete.
- Scaling smaller parts of an application results into scaling larger part of the application as a part can't be scaled independent of another part of an application.
- Multiple teams are to be involved to make a change in one of the features and that becomes time consuming.
- Testing entire monolith application (after any changes) increases a risk due to its complexity and eventually results into slow delivery resulting into longer turnaround times for releases.

Principles of Micro services and hence Micro frontends

Principles of microservices that are inherently applicable to micro frontends are:

High Cohesion/Low Coupling	High Cohesion/ Low Coupling illustrates a single responsibility, it should focus on specific domain or subdomain, should be autonomous that demonstrates Independently deployable or changeable unit without affecting any other part. It should be resilient to failure (without impacting other micro frontends), should be transparent in terms of health of each of the components, and should be testable with automation tools. Furthermore, two micro frontends should not depend on each other, else they make a case for combining into a single micro front end application.
Independent Team	Each micro frontend (offering distinct feature or business domain or business function offering value to business) is owned by a team end to end and a base application can be owned by a dedicated team whereas entire team of a micro frontend should be cross functional. This improves agility of an organization and eases communication within the team.
Technology Agnostic	Each micro frontend can be technology agnostic resulting into overall application composed of multiple frameworks, one per micro frontend. This helps in the cases when any specific technology/ framework becomes obsolete or requires upgrade. In that case, individual micro frontend can be upgraded or re-architected independently without causing technology debt due to delays in upgrade/ re-architecture of a complete application based on a single technology/ framework. It also helps each application to select right technology based on its need than sticking to a common technology when lightweight framework can serve the purpose at times for a specific application.

User Experience	All micro frontend applications should offer similar user experience (theming, style etc.) and performance. Hence architect is suggested to be mindful while using heavy/ high number of frameworks resulting into performance issues and base application should offer a style guide for micro applications. Additionally, user interface should be responsive and multiple parts of an application should be updated upon user action, even though they belong to different micro frontend applications. Furthermore, an application composed of micro frontend applications should work consistently across all browsers/ versions.
Microservices Driven	Micro frontends should be micro services driven to support its functionality resulting into modular frond end and backend as well.

Where to leverage Micro frontends

Multiple Teams	A team is comprised of multiple sub-teams and each sub-team specializes only in specific domain or subdomain pertaining to the feature that they deliver.
Reduced Complexity	A reduction in application complexity is to be achieved through decoupling of subset of application (s) from each other through division of application into multiple independent applications that form an overall application.
Multiple Technology	One or more features of an application are planned to be built on different technology/ framework.
Technology Upgrade	Technology upgrade of subset of an application at a time is anticipated due to technology debt/ challenges within an organization.
Independent Deployments	Each feature of an application is expected to have its own release cycles, independent of each other.
High Resiliency	High resiliency is one of the key requirements of an application, that can be achieved through individual micro apps distributed in small chunk of services (covering all layers) independently. This also offers an advantage of only stipulated portion of an application having an impact as opposed to the complete application.

Approaches to implement Micro Frontends

Micro frontends can be implemented through various approaches as following:

Approach for isolated micro-apps:

Micro applications that aren't planned to be on the same runtime and hence are said to run in complete isolation, can adopt one of the following approaches:

- **Hyperlink or URL based Micro apps:**

  Independent micro frontend applications are linked through a dashboard of an application through hyperlinks/ URLs to demonstrate illusion of them being a single application as navigation is offered by a common dashboard. All micro applications use common CDN and UI components/ guidelines for uniformity. Additionally, all micro applications should use common authentication method to avoid authenticating for each application separately.

  Adherence to principles:

  - **High Cohesion/ Low Coupling:** Hyperlink/ URL based approach adheres to the principles of high cohesion and individual applications are testable or changeable. Also, they can communicate through query string parameters or micro services can share data behind the scenes when involved through a micro application.
  - **Separate team owing each micro application:** This principle is also met as each app is a separate application.
  - Technology agnostic: Each micro app can be developed using any technology and hence this principle is met as well.
  - **Common user Experience:** This principle is met to an extend with common styles, but user interface flow can have limitations and hence can result into poor user experience as no two micro applications can be opened at the same time or change in one can't render related use case from another application in the same user interface screen/ page.
  - **Feature driven:** As each micro app is separate, it can offer distinct business value in the form of a specific business feature and hence hyperlink/ URL based micro app approach meets this principle.
  - **Micro services driven:** Every micro app can be driven by backend micro services and hence there is no limitation with respect to this principle using hyperlink/ URL based approach.

- **iFrames based:**

  Multiple Micro applications are rendered through iFrames within user interface of an application that provides ability to have all applications in one web page/ screen through base/ shell application, hence offer better uniformity and business flows.

  Adherence to principles:

  Adherence to principles that deviate from how approach of hyperlink or URL based micro apps adhere to micro frontend principles is covered below. Anything that is not covered following is same as adherence details covered under hyperlink or URL based micro apps:

  - High Cohesion/ Low coupling: iFrame based approach adheres to the principles of high cohesion and individual applications are testable or changeable. Also, they can communicate through events or background micro services can share data for micro frontend enable communication.
  - Common user Experience: This principle is met to an extend with common styles and user interface flow can also be rendered seamlessly to provide greater user experience as multiple/ all micro applications are generally rendered within separate iFrames on the same web page resulting into updates to related use case from another application reflected in the same user interface screen/ page. But performance may become an issue resulting into slightly poor user experience due to each application based on separate runtime and/ or JS frameworks.
  - Feature driven: As each micro app is separate, it can offer distinct business value in the form of a specific business feature and this principle is better met than hyperlink/ URL based approach since multiple applications are rendered on the same user interface offering integrated functional/ business value provided by multiple apps together.

Approach for micro-apps on common runtime:

If all Micro applications are planned to be based on a common runtime, then one of the following approaches can be adopted for a micro frontend:

- **WebComponents based:**

  Each micro app is rendered on a web page through a web component (feature of a web-based technology) to form micro frontend. This results into memory and resources being shared by all micro applications. A WebComponent is made up of custom HTML element and is included in a web page through HTML imports. Web components aren't tightly coupled with a JS framework but are formed by HTML, CSS and JS.

  WebComponents meet all principles of micro frontends. Communication between micro apps is handled through events, and no issue related to user interface are seen with WebComponents as all WebComponents are a part of parent DOM and hence sizing related issue with respect to individual applications aren't observed as it's a possibility with iFrame based approach, but performance aspects should be balanced with limited number of frameworks used by various micro apps.

Web Components allows creation of custom and reusable HTML tags that can be used in web pages which works across modern browsers. Following three technologies are used together to create Web Components:

Custom Elements:

Custom elements give developers the ability to extend HTML and create their own tags. Because custom elements are standards based, they benefit from the Web's built-in component model. The result is more modular code that can be reused in many different contexts.

Shadow DOM:

Shadow DOM is a web standard that offers component style and markup encapsulation. It is a critically important piece of the Web Components as it ensures that a component will work in any environment even if other CSS or JavaScript is at play on the page.

HTML Templates:

With HTML Templates, we create HTML fragments which remain inactive and unrendered until explicitly requested. These can then be reused multiple times as the basis of a custom element's structure.

Please check Appendix section to Deep Dive into WebComponents using Angular Elements.

- **JS Framework based:**

  JS framework-based approach adapted for the base/ shell application leads to additional cost, and it gets difficult to move away from that framework over the years resulting into difficulty to migrate. Hence this approach should be carefully selected while implementing micro frontends.

  Examples of JS Frameworks available in the market to implement micro frontends are TailorJS, Single-spa, Puzzle.JS etc.

  JS framework-based approach meets all principles of micro frontends, except:

  - **High Cohesion/ Low Coupling:** If Base/ shell application and one or more micro applications are based on a common JS framework, then changes in one micro application results into changes in the base application even though each micro app is built for specific business features. This is due to technology specific constructs that base app/ micro apps are based on. And hence doesn't abide by high cohesion/ low coupling principle.
  - **Technology Agnostic:** JS framework-based approach locks into a specific technology framework, especially when it's a strategic decision to invest in only one or limited number of frameworks is chosen to avoid getting into performance issues impacting user experience and hence isn't considered as technology agnostic implementation.

- **Webpack 5 Module Federation:**

  Module federation allows a JavaScript application to dynamically run code from another bundle/build, on client and server. Module Federation is integrated in Webpack with version 5 and is one of the emerging and could be said as official solution for implementation of micro frontends. Terminologies are explained below:

  - **Module Federation:** loads the code from another application.
  - **Host:** A webpack build that is initialized first during a page load (when the onLoad event is triggered), we could also call it Shell.
  - **Remote:** Another Webpack build, where part of it is being consumed by a "host".
  - **Bidirectional-hosts:** when a bundle or Webpack build can work as a host or as a remote. Either consuming other applications or being consumed

Comparison of Micro Frontend Approaches

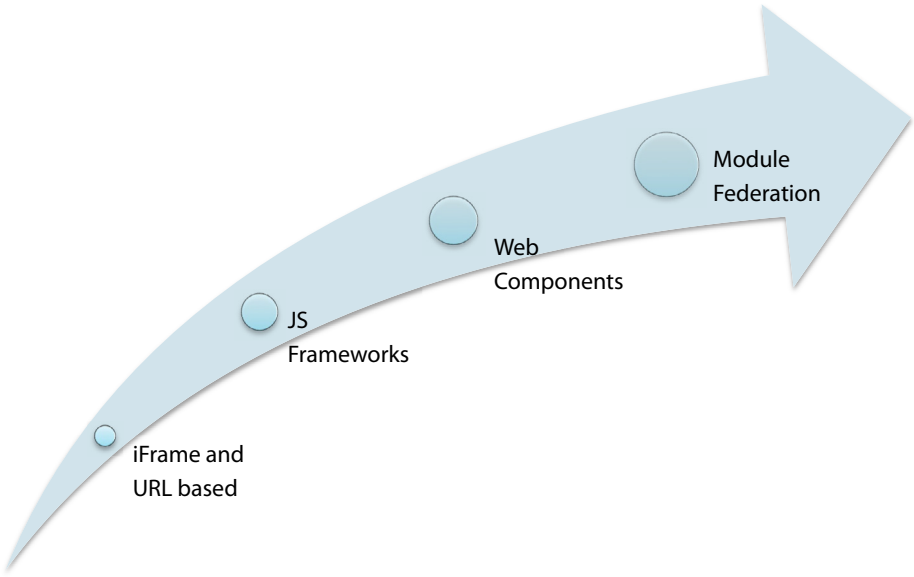
Comparison of various approach of micro frontend implementation is covered as following:

Approach/ Criteria	Dev Complexity/ Learning Curve	Deployment Complexity	Support for Base app	Support for Tree shaking	Ability to share Dependencies
Hyperlink/ URL based	Low	Low	No	Yes	No
iFrames based	Low	Low	No	Yes	No
WebComponent based	High	Medium	Yes	Yes	No
JS Framework -> TailorJS	High	High	No	No	No
JS Framework -> Single-spa	High	High	Yes	No	Yes
Webpack 5 Module Federation	High	High	Yes	Yes	Yes

Infrastructure cost is similar for all the approach mentioned above, whereas they all embrace the principle of independent development quite effectively, and hence each micro application can be deployed independently. They all support multiple frameworks/ multiple versions of the framework in a micro frontend implementation and none of them are convenient to bundle together.

Maturity Model of Micro frontends

Micro frontends first came up in ThoughtWorks Technology Radar around 2016. Before coining this term, applications were developed using iFrames which could be said as earliest form of Micro frontends, and with Webpack 5 Module Federation, Micro frontends concept seems to be officially accepted and has matured. Micro fronts is still an emerging and being adopted.



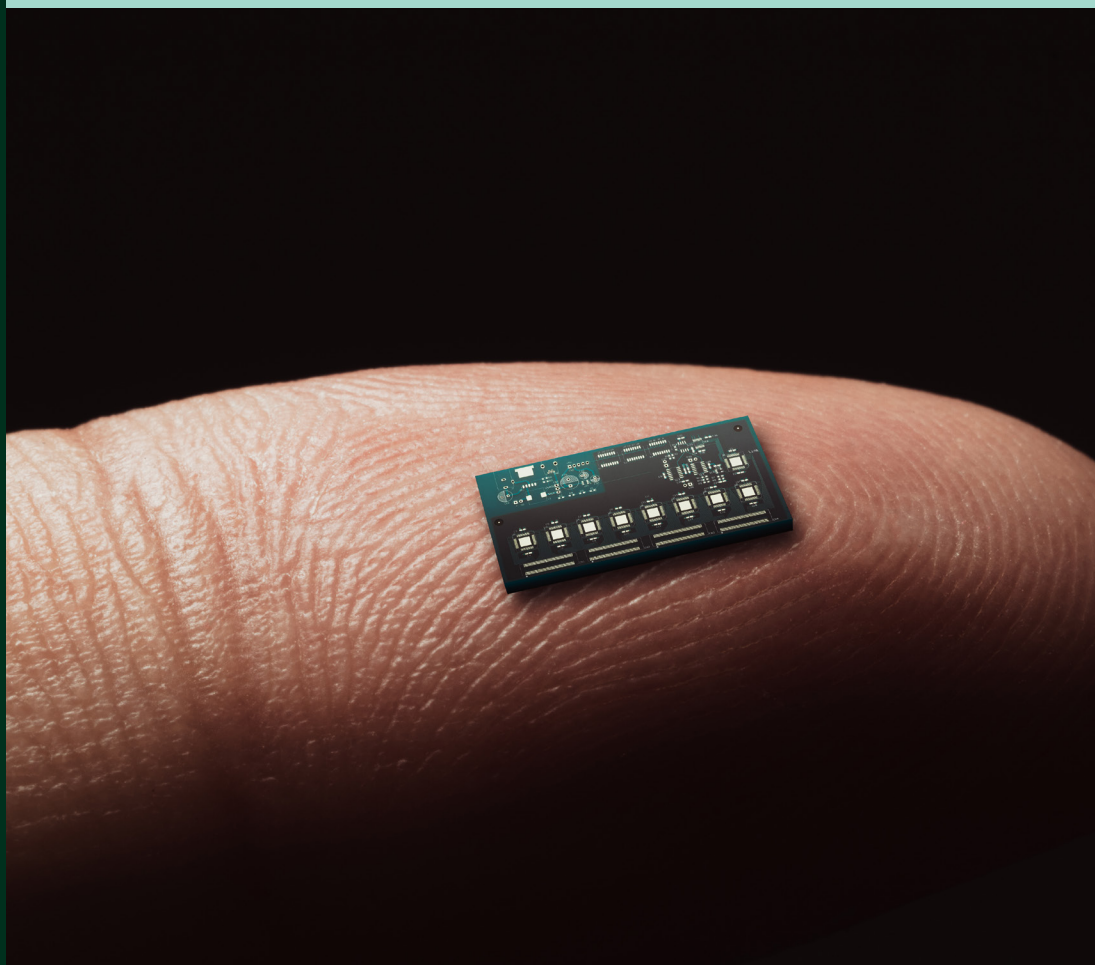
Summary

Based on application requirements, micro frontends can be implemented with either vertical or horizontal split approach. If there aren't complex user interface flows that require multiple view and hence micro frontends to exist in the same view, vertical split is preferred approach as it reduces complexity to the great extent. And in this scenario, Hyperlink or URL based approach/ iFrame based approach works well. But if an application mandates multiple micro application to work together in the same view to offer greater user experience, then horizontal split is the only available option to implement micro frontends. Horizontal split can be accomplished through Webpack 5 Module Federation, WebComponents or any JS based frameworks that offer a capability to build micro frontends. Micro frontend is an evolving architectural style and hence more open-source JS based frameworks are coming up in the market. While each one of them target at offering micro frontend capabilities, apart from basic decisions covered in this paper, a choice of framework for implementation purely depends on organizational standards when it comes to adoption of a specific approach/ framework.

About the Author

Pabitra Mohan
Technology Architect

Trupti Premanand Patil
Senior Technology Architect



Appendix: Deep Dive into Web Component based Micro Frontend Approach

As Angular provides Angular Elements (internally through custom elements) as a feature that offers a capability to create new DOM elements, demonstration of implementation using Angular is as covered following:

- **Create a new workspace:**

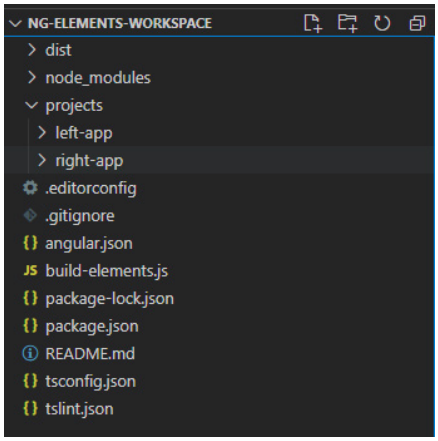
  ng new ng-elements-workspace --createApplication=”false”

  Create both Angular application which will be converted to Angular Elements

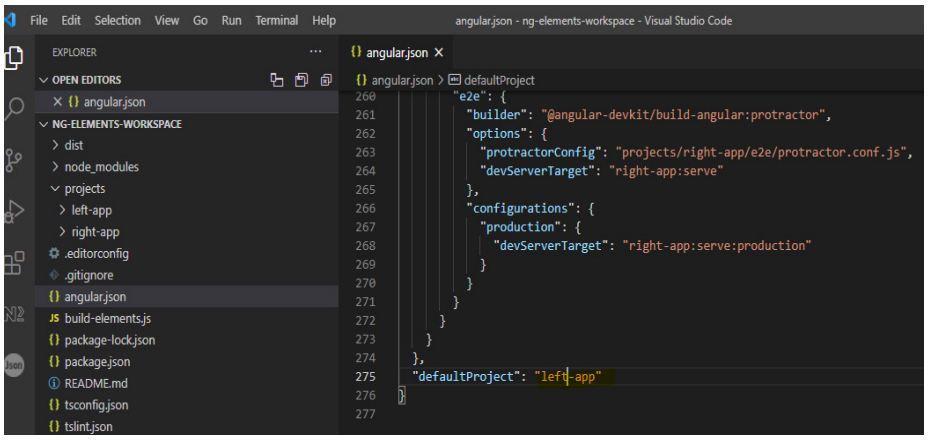
  ng generate application left-app

  ng generate application right-app

  The project structure will be as following:



- **Adding Angular Elements:**
  - In the root workspace folder set the defaultProject to left-app



- **Add Angular Elements**

  ng add @angular/elements

- **Update Index.html with new tags**

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>LeftApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <left-app></left-app>
</body>
</html>
```

- **Update Main.js to get rid of Zone.js**

```
platformBrowserDynamic().bootstrapModule(AppModule, {ngZone: 'noop'})
  .catch(err => console.error(err));
```

- **Update app.module.ts**

  Remove AppComponent from Bootstrap and add entryComponent having AppComponent

  Create Constructor initializing Injector

  Create customElement and define the tag (left-app)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Injector } from '@angular/core';
import { createCustomElement } from '@angular/elements';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [],
  entryComponents: [
    AppComponent
  ]
})
export class AppModule {
  constructor(private injector: Injector) {}
  ngDoBootstrap() {
    const leftApp = createCustomElement(AppComponent, { injector: this.injector });
    customElements.define(left-app, leftApp);
  }
}
```

- **Update App.component.html**

  Create a simple html, here the html will show username and role as provided to the Web Component in Input Parameters.

  Additionally, add a button that will pass information to other Angular Element (right-app) when its clicked.

```
<h1>User: {{ username }}</h1>
<br>
<h2>Role: {{ role }}</h2>
<button (click)='send_right()'>Update quantity to 10</button>
```

- **Update App.component.ts**

  Add Encapsulation and the @Input for username and role.

```
import { Component, ViewEncapsulation, Input } from '@angular/core';

@Component({
  selector: 'app-um',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.ShadowDom
})
export class AppComponent {
   @Input() username = 'test username';
  @Input() role = 'test role';
  title = 'left-app';

  send_right() {
    const data = {
action: '10'
    };

    const event = new CustomEvent('ceLeftApp', {detail: data});
    window.dispatchEvent(event);
  }
}
```

- **Repeat the above steps by changing the defaultProject to "right-app"**

  App.component.html (right-app):

```
<h1>Product: {{ product }}</h1>
<br>
<h2>Quantity: {{ quantity }}</h2>
```

App-component.ts (right-app)

Event Listener is used to listen to events from left-app

```
import { ChangeDetectionStrategy, ChangeDetectorRef, Component, Input, OnDestroy, OnInit,
ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-right',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush,
  encapsulation: ViewEncapsulation.ShadowDom
})
export class AppComponent implements OnInit, OnDestroy {
  @Input() product = 'test product';
  @Input() quantity = 'test quantity';
  title = 'right-app';

  constructor(private cd: ChangeDetectorRef) {
  }

  // tslint:disable-next-line:typedef
  ngOnInit() {
    window.addEventListener('ceLeftApp', this.customEventListenerFunction.bind(this), true)
;
  }

  // tslint:disable-next-line:typedef
  customEventListenerFunction(event) {
    this.quantity = event.detail.action;
    this.cd.detectChanges();
  }
  ngOnDestroy(): void {
    window.removeEventListener('ceLeftApp', this.customEventListenerFunction, true);
  }
}
```

With this, it will update quantity when the button on the left-app is clicked.

- **Bundling Angular Elements**
  - Repeat the below steps with defaultProject set to left-app and then to right-app
  - Using ngx-build-plus

Ngx-build-plus provides necessary tools to compile the Angular Application to Web Components.

Run the following to add ngx-build-plus

- ng add ngx-build-plus
- ng g ngx-build-plus:wc-polyfill: Adds webcomponent polyfills to your app
- ng g ngx-build-plus:externals: Updates your app to use webpack externals

- Install fs-extra and concat, which helps to bundle the files.
- npm i fs-extra –save-dev
- npm i concat –save-dev
- Create bundle-element.js in root folder

```
const fs = require('fs-extra');
const concat = require('concat');
(async function build() {
  const prgName = process.argv.slice(2)[0];
  if (prgName === '' || prgName === undefined) {
    console.log('Project name is required');
  } else {
  const files_es2015 = [
    './dist/' + prgName +  '/polyfill-webcomp-es5.js',
    './dist/' + prgName +  '/polyfill-webcomp.js',
    './dist/' + prgName +  '/polyfills.js',
    './dist/' + prgName +  '/scripts.js',
    './dist/' + prgName +  '/main.js'
  ];
  await fs.ensureDir('./dist/' + prgName + '/elements');
  await concat(files_es2015, './dist/' + prgName +  '/elements/' + prgName + '.js');
  console.log('Done generating bundle for ' + prgName);
}
})();
```

- **Update package.json file by adding --output-hashing none && node build-element.js in build:left-app:externals and build:right-app:externals**

  "build:left-app:externals": "ng build --extra-webpack-config projects/left-app//webpack.externals.js --prod --project left-app --single-bundle --output-hashing none && node build-elements.js left-app",

  "build:right-app:externals": "ng build --extra-webpack-config projects/right-app//webpack.externals.js --prod --project right-app --single-bundle --output-hashing none && node build-elements.js right-app"

- **Build the code by using following command:**

  npm run build:left-app:externals

  npm run build:right-app:externals

  Once successful elements folder would be created in dist/left-app and dist/right-app with bundled Javascript files left-app.js and right-app.js

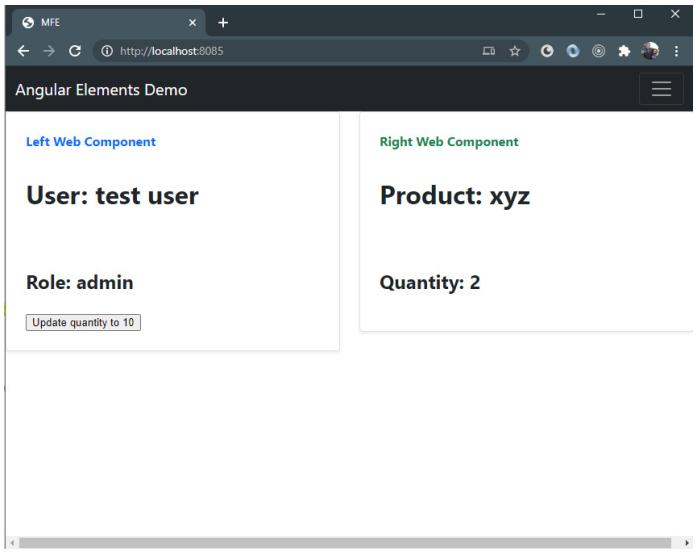
  Copy these files to another folder say MFEDemo

  Create an index.html file with following content:

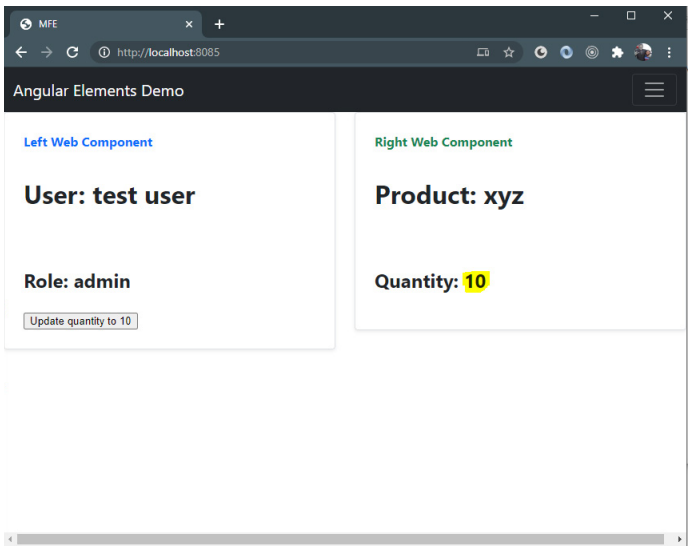
```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
  <base href="/">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1" crossorigin="anonymous">
    <title>MFE</title>
  </head>
  <body>
  <main>
  <nav class="navbar navbar-dark bg-dark" aria-label="First navbar example">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Angular Elements Demo</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarsExample01" aria-controls="navbarsExample01" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
    </div>
  </nav>

<div class="row mb-2">
    <div class="col-md-6">
      <div class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-
250 position-relative">
        <div class="col p-4 d-flex flex-column position-static">
          <strong class="d-inline-block mb-2 text-primary">Left Web Component</strong>
          <left-app username="test user" role="admin"></left-app>

        </div>
        <div class="col-auto d-none d-lg-block">
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <div class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm h-md-
250 position-relative">
        <div class="col p-4 d-flex flex-column position-static">
          <strong class="d-inline-block mb-2 text-success">Right Web Component</strong>
          <right-app product="xyz" quantity="2"></right-app>
        </div>
        <div class="col-auto d-none d-lg-block">
        </div>
      </div>
    </div>
  </div>
  </main>
    <!-- Optional JavaScript; choose one of the two! -->
    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW" crossorigin="anonymous"><
/script>
  <!---->
  <script type="text/javascript" src="left-app.js"></script>
  <script type="text/javascript" src="right-app.js"></script>
  </body>
</html>
```

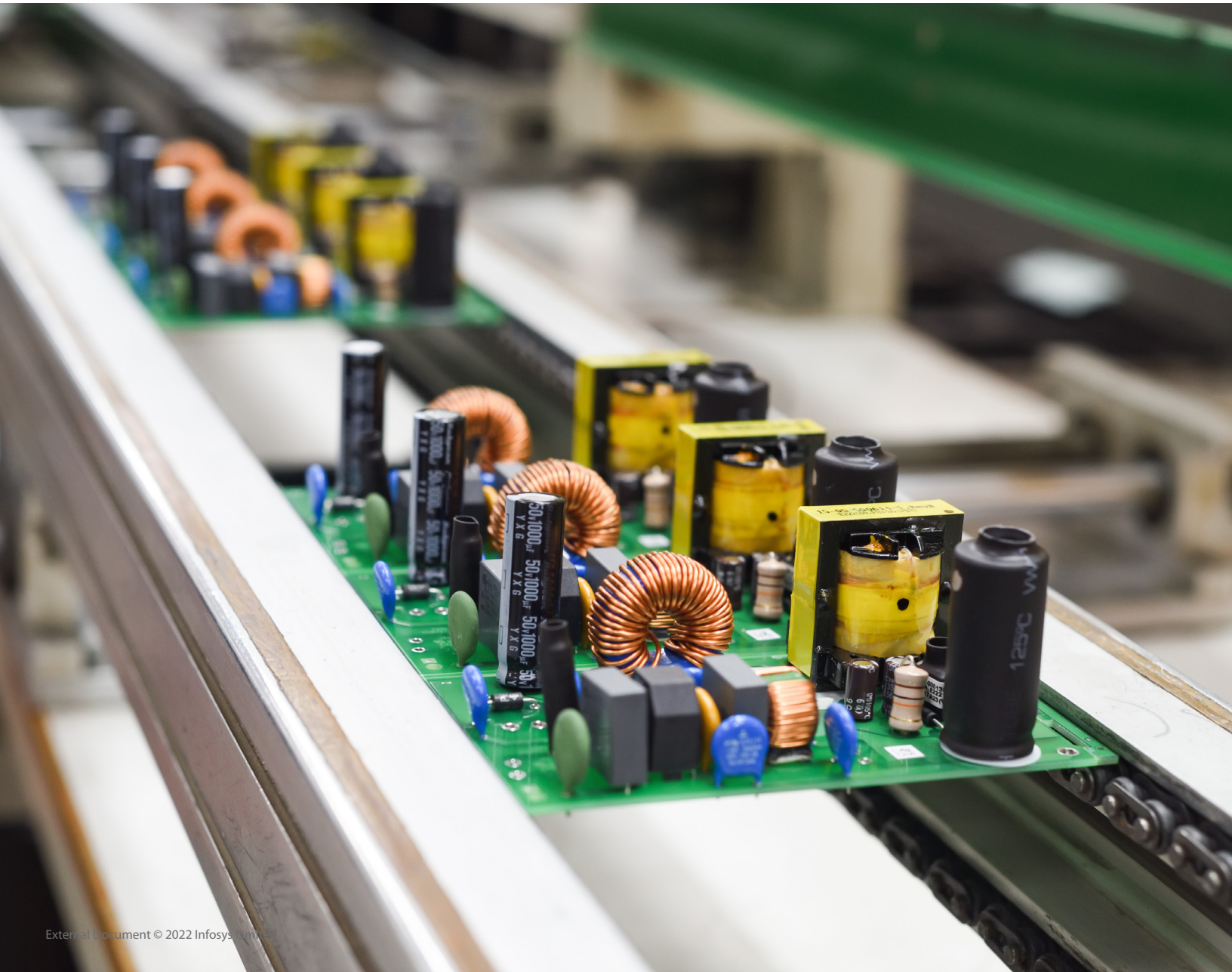
- **Run the index.html file using http-server**

  http-server -p 8085



- **On button click, 2nd web-component will be updated as following:**





References

https://www.thoughtworks.com/radar/techniques/micro-frontends

https://martinfowler.com/articles/micro-frontends.html

https://micro-frontends.org/

https://dev.to/phodal/micro-frontend-architecture-in-action-4n60

https://www.angulararchitects.io/en/aktuelles/6-steps-to-your-angular-based-microfrontend-shell/

https://www.angulararchitects.io/en/aktuelles/a-software-architects-approach-towards/

https://single-spa.js.org/

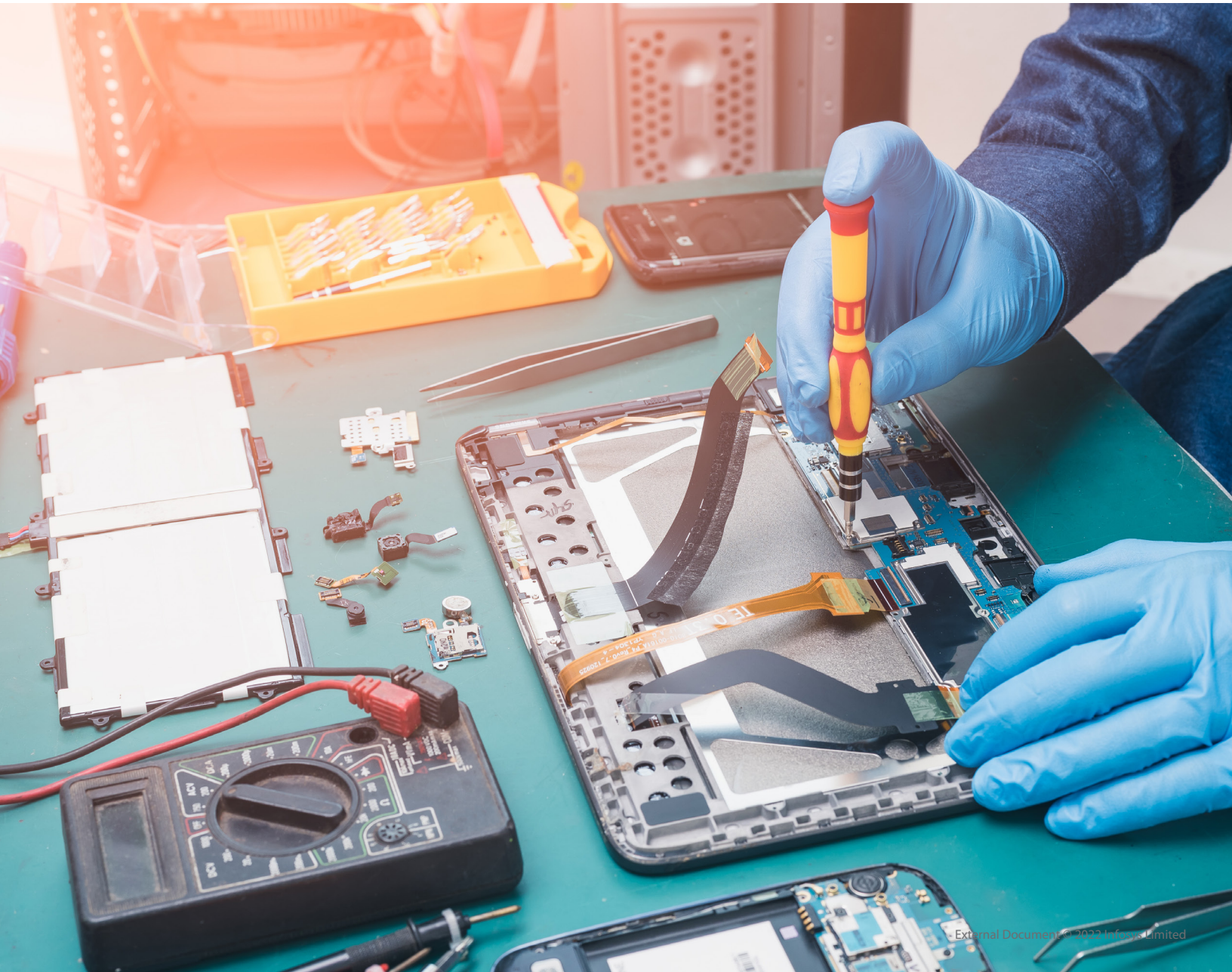
https://developer.mozilla.org/en-US/docs/Web/Web_Components

https://webpack.js.org/concepts/module-federation/

https://medium.com/swlh/webpack-5-module-federation-a-game-changer-to-javascript-architecture-bcdd30e02669

https://medium.com/@collin6308/angular-elements-steps-a65fef734999

https://github.com/akhil110/microfrontend_workspace



For more information, contact askus@infosys.com

Infosys®
Navigate your next