

# Report: Micro-frontends

## Summary:

This article offers a concise overview of micro-frontend architecture, detailing its definition and operational principles. It outlines the core concepts driving this architectural approach and provides a comprehensive guide on its implementation using web components. Furthermore, the article offers practical insights by demonstrating the creation of a basic e-commerce platform, serving as a hands-on example for readers. Additionally, it includes a codebase for the project. Unfortunately it does not delve deeper into common issues such as routing, shared dependencies, and more.

## Keywords:

MF implementation

MF concepts

Web components

## Research Questions:

1. What are Micro Frontends?
2. How to implement micro-frontends using web components?

## Key Points:

### → Main idea behind micro-frontends

- Consider a web application as a collection of features managed by separate teams.
- Each team specialises in a distinct area of business
- A team is cross functional and develops its features end-to-end, from database to user interface. [1]

### → Core Ideas behind Micro-Frontends

- Each team should be able to choose and upgrade their stack without having to coordinate with other teams.
- Construct independent applications that are self-contained.
- Establish naming conventions in areas where complete isolation is not feasible yet.
- Prioritise native browser features over custom.
- Ensure that your feature remains functional and useful even if JavaScript fails or hasn't executed yet. [3]

### → Web components (Custom Element)

- extension of HTMLElement
- name must include a dash (-) to maintain compatibility with upcoming new HTML tags
- naming convention [team\_color]-[feature] guards against collisions and the ownership of a feature becomes obvious [6]

### → Browser Support

- the “Custom Element V1 Spec” is supported by all modern browsers.
- No workarounds are needed. [8]

### → Framework Compatibility

- Custom Elements are a web standard, all major JavaScript frameworks (such as React/Vue/Angular/Svelte/Preact) support them
- These frameworks offer methods for releasing framework-specific application as a Custom Element
- Allow to embed a Custom Element just like a native HTML tag [8]

### → Child-Parent or Siblings Communication / DOM Events

- Use a PubSub mechanism instead of internal JavaScript API, where a component can publish a message and other components can subscribe to specific topics
- Browsers have this feature built-in
- Possibility to create higher level events with new CustomEvent(...) [9]

### → Server-side Rendering / Universal Rendering

- Each team has their own express server and the render() method of the Custom Element is also accessible via url.
- The Custom Element tag name is used as the path name – attributes become query parameters. [11]

### → Data Fetching & Loading States

- A downside of the SSI/ESI = the slowest fragment determines the response time of the whole page
- Response of a fragment should be cached
- Omit fragments that are resource-intensive to generate and difficult to cache, from the initial rendering. Instead, they can be asynchronously loaded in the browser.
- Use a technique Skeleton Screens to solve substantial reflow issue [14]

## Infographics:

### End-to-End Teams with Micro Frontends

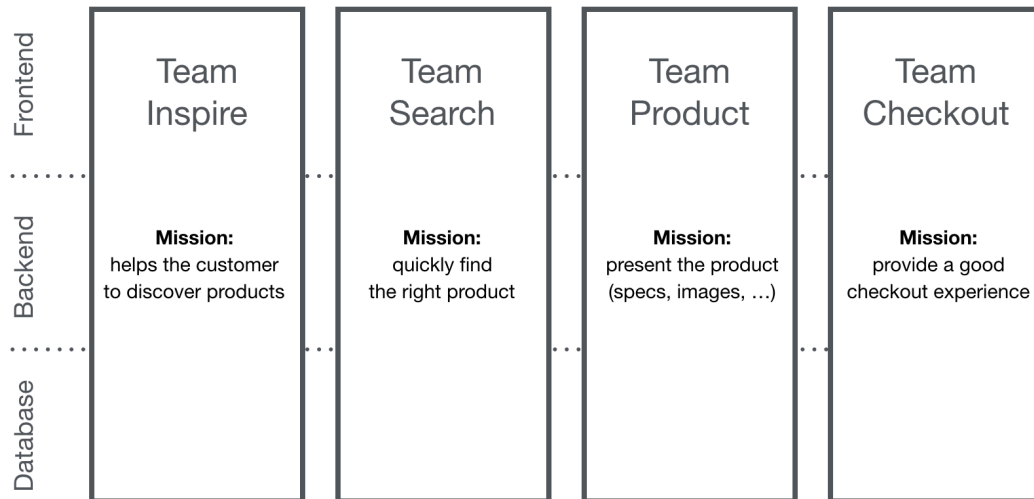


Table. 1: Organisation in Verticals [2]