**Introduction to Micro Frontends**

In this lesson, we will understand micro frontends.

**We'll cover the following**

* + [What are micro frontends?](https://www.educative.io/module/lesson/web-application-architecture-101/g758262jyGY#What-are-micro-frontends?)
  + [Micro frontends e-commerce application example](https://www.educative.io/module/lesson/web-application-architecture-101/g758262jyGY#Micro-frontends-e-commerce-application-example)

**What are micro frontends?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/g758262jyGY#What-are-micro-frontends?)

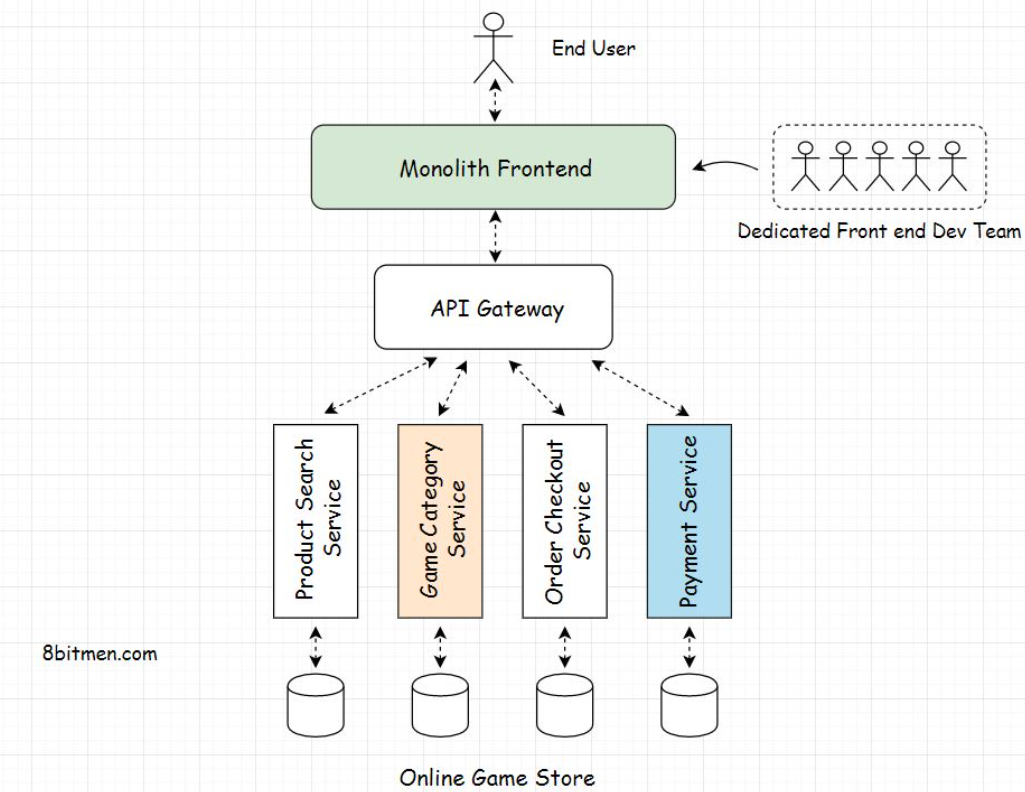
*Micro frontends* are distinct loosely coupled components of an application’s user interface developed applying the concept of *microservices* to the frontend.

Writing *micro frontends* is more of an architectural design decision and a development approach as opposed to it being a technology.

What does applying the concept of microservices to the front end mean?

*Microservices* provide complete autonomy to the teams developing them. They are loosely coupled, provide fault isolation, and offer the freedom to pick the desired technology stack to the dedicated teams, to develop a certain service. Micro frontends offer the same upsides to frontend development.

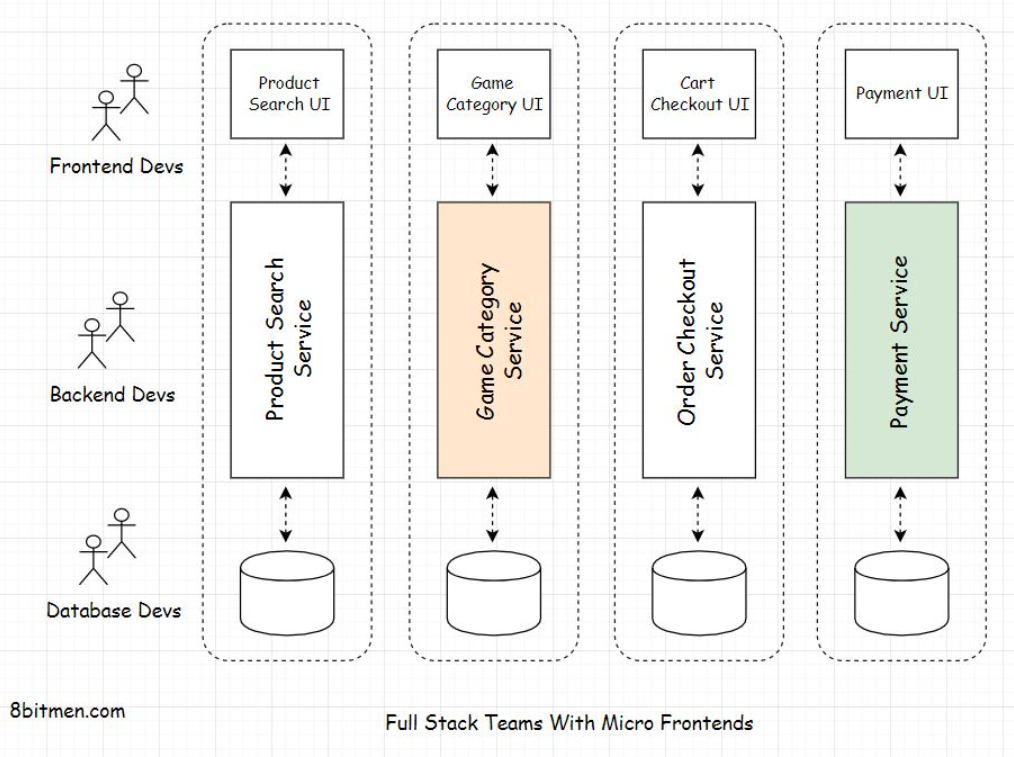
Typically, in application architecture, despite having microservices on the backend, our frontend is a monolith that is developed by a dedicated frontend development team.



With the micro frontends approach, we split our application into vertical slices, where a single slice goes end to end right from the user interface to the database. And every slice is owned by a dedicated team.

Besides having the backend devs, the micro frontend team of a particular vertical slice also includes the frontend developers who develop the user interface component only for that specific service.

Every team builds their user interface component choosing their desired technology, and later all these components are integrated, forming the complete user interface of the application. This micro frontend approach averts the need for a dedicated centralized user interface team. Every micro frontend team becomes more of a full-stack team.



Let’s break this down further with the help of an example.

## Micro frontends e-commerce application example[#](https://www.educative.io/module/lesson/web-application-architecture-101/g758262jyGY#Micro-frontends-e-commerce-application-example)

I’ve picked the example of an e-commerce application because the micro frontends approach is pretty popular with e-commerce websites.

Imagine an online game store that delivers video games for desktops and consoles such as Xbox, Nintendo Switch, PlayStation, and the related hardware.

Our online gaming store will have several different UI components. A few of the key components would be:

**The search component** – This is a search bar at the top center of the website that enables the users to search games based on the keywords they enter.

Once the user runs a search, the component enables the user to filter their search results based on several options, including the price range, type of console, game genre, and so on.

**The game category component** – This component displays the popular and widely searched games for different categories on the website’s homepage.

**Add to cart and checkout component** – This user interface component enables users to add games of their liking to their cart and proceed to the checkout filling in their address and other required information to make the final payment.

During the checkout, the website may also recommend related games to the user as upsells. The user can also feed in coupons and gift cards if they have any.

**The payment component** – The payment UI component offers different payment options to the user and facilitates the order payment once the user enters their card details on the page.

Every UI component has a dedicated microservice running on the backend powering that particular user interface component. All these different components are developed and managed by dedicated full-stack (micro frontend) teams.

The application’s complete user interface is rendered combining all these different UI components, also called micro frontends.

Let’s continue this discussion in the next lesson.

**The need For Micro Frontends**

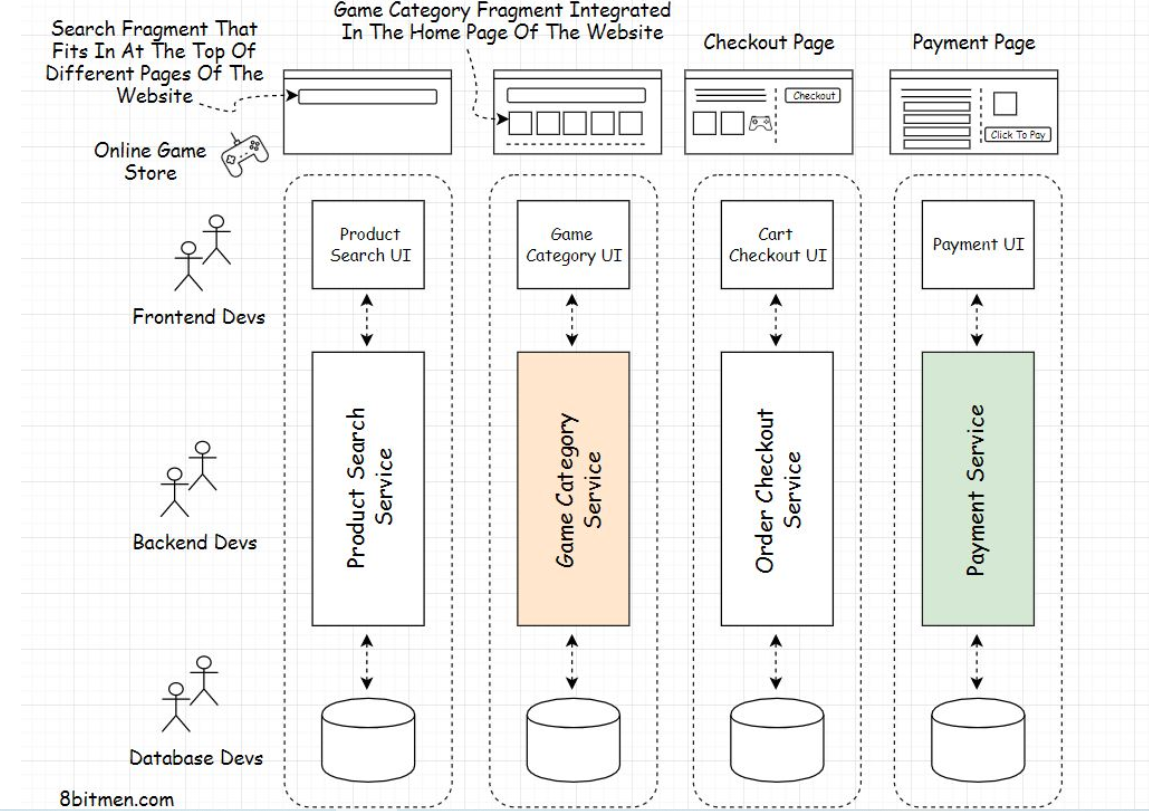
This lesson continues the discussion on micro frontends from the previous lesson.

**We'll cover the following**

* + [Easier coordination between the frontend and the backend devs](https://www.educative.io/module/lesson/web-application-architecture-101/xVpmrq1Ro33#Easier-coordination-between-the-frontend-and-the-backend-devs)
  + [Leveraging the right technology](https://www.educative.io/module/lesson/web-application-architecture-101/xVpmrq1Ro33#Leveraging-the-right-technology)

A micro frontend team may own a more extensive UI component like the *checkout page*. They may own a minor component that fits in a particular component like the *game category* component on the *home page*. Or they might own both.

The smaller components that integrate into other pages/components of the application are known as *fragments*.



Now, you might be wondering, what are the significant upsides of splitting an application into micro frontends?

Let’s find out.

## Easier coordination between the frontend and the backend devs[#](https://www.educative.io/module/lesson/web-application-architecture-101/xVpmrq1Ro33#Easier-coordination-between-the-frontend-and-the-backend-devs)

When we have full-stack teams owning an entire service end to end, this averts the need for a dedicated frontend team. In addition, since the frontend devs now work alongside the backend devs on the same team, this saves a lot of time that was initially spent in the cross-team coordination.

With this new approach, communication is quick and not so formal. This improves the team’s productivity and enables them to deliver a better user experience by having more effective coordination between the backend and the frontend devs.

## Leveraging the right technology[#](https://www.educative.io/module/lesson/web-application-architecture-101/xVpmrq1Ro33#Leveraging-the-right-technology)

Since the micro frontends are loosely coupled, we can develop them leveraging different technologies, just like microservices. This opens up our options as opposed to sticking to just one UI technology to build the complete front end of the website.

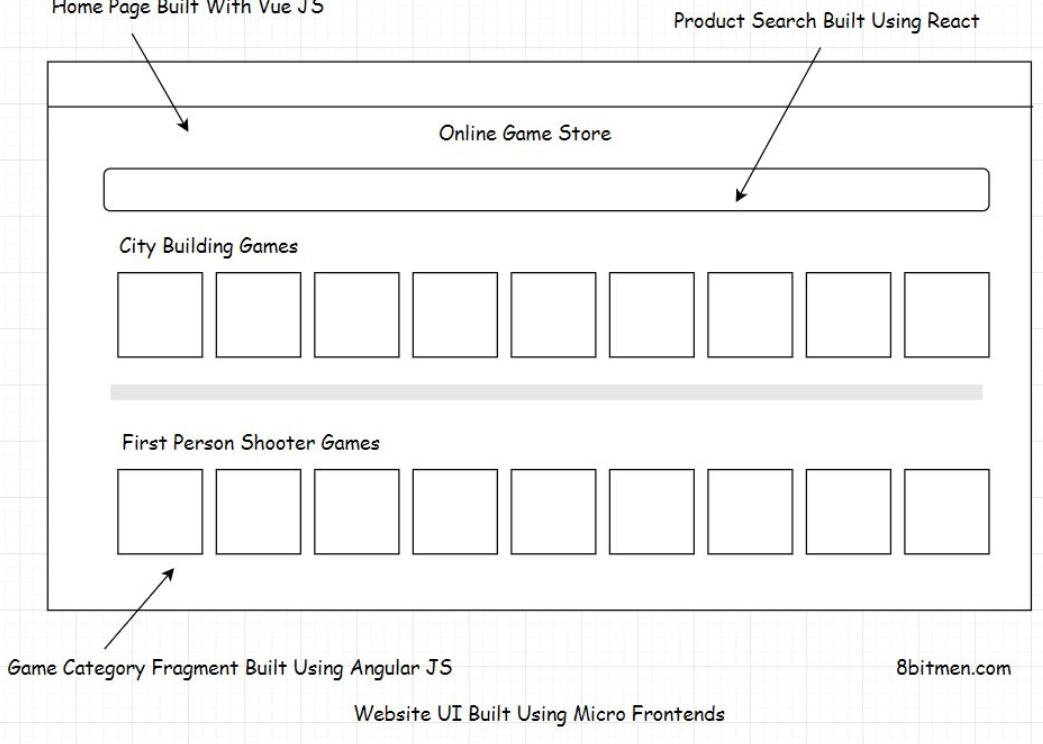
There are a plethora of existing frontend technologies in the industry that cater to different use cases, in addition to the new waves of JavaScript frameworks that hit us every year.

With the micro frontends approach, we can pick the right technology to build our frontend components. We often have use cases where just plain JavaScript, HTML, and CSS suffice to build a feature, and then there are other cases where we need advanced frameworks like React, Angular, and Vue to implement our features.

With micro frontends, we don’t necessarily have to use React or a similar framework to build a certain component if we don’t need it, despite having other components being implemented using it.

On the flip side, if our website is built on plain JavaScript and we want to use React, we don’t have to re-write the entire website to use React. We can just write specific components that need React and integrate them into the website.

Even if multiple teams use the same technology to build their UI components, they can work on different versions of the technology. They can easily upgrade their libraries without impacting the website’s other UI components.



Now, moving forward with the micro frontends approach may sound delightful, but it’s only fit for medium to large websites. This approach won’t be that advantageous for simple use cases. Rather, it will make things more complex and prove to be overkill.

Using multiple technologies in a project brings along a lot of architectural and maintenance complexities with it.

With micro frontends, we need to write additional code to combine all the components built with heterogeneous tech. We also have to deal with compatibility and performance issues when using multiple technologies together. So, there are always trade-offs involved. There is no silver bullet.

**Here is an interesting watch** – [Engineering culture at Spotify](https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/) (<https://engineering.atspotify.com/2014/03/spotify-engineering-culture-part-1/>)

In the next lesson, let’s have a quick insight into how these micro frontends are integrated with each other.

**Micro Frontends Integration**

In this lesson, we will discuss how micro frontends are integrated with each other to make the complete UI of a website.

**We'll cover the following**

* + [Client-side integration of micro frontends](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Client-side-integration-of-micro-frontends)
  + [Server side integration](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Server-side-integration)
  + [Technology and frameworks facilitating server-side integration](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Technology-and-frameworks-facilitating-server-side-integration)

Once we have respective micro frontends ready for our online game store, we need to integrate them together to have a functional website. There are two ways we can do this -

1. By integrating micro frontends on the client
2. By integrating micro frontends on the server

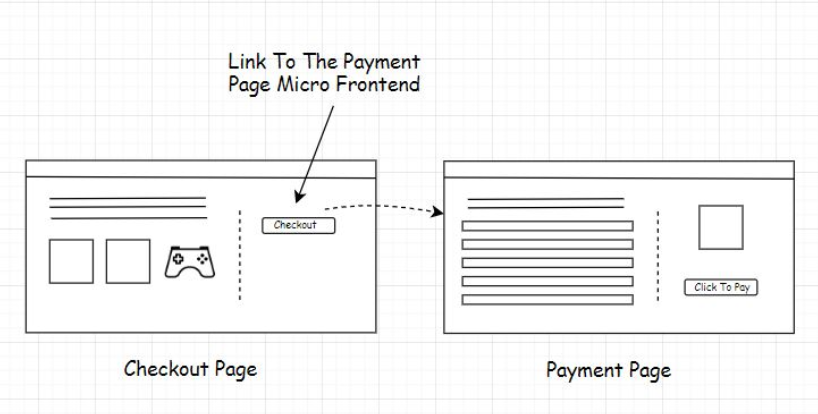
This concept is similar to the *client-side* and *server-side rendering*, which we discussed earlier in the course.

In this micro frontend scenario, we need to write additional logic to enable the integration of the UI components.

Let’s discuss the *client-side integration* process first.

**Client-side integration of micro frontends**[#](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Client-side-integration-of-micro-frontends)

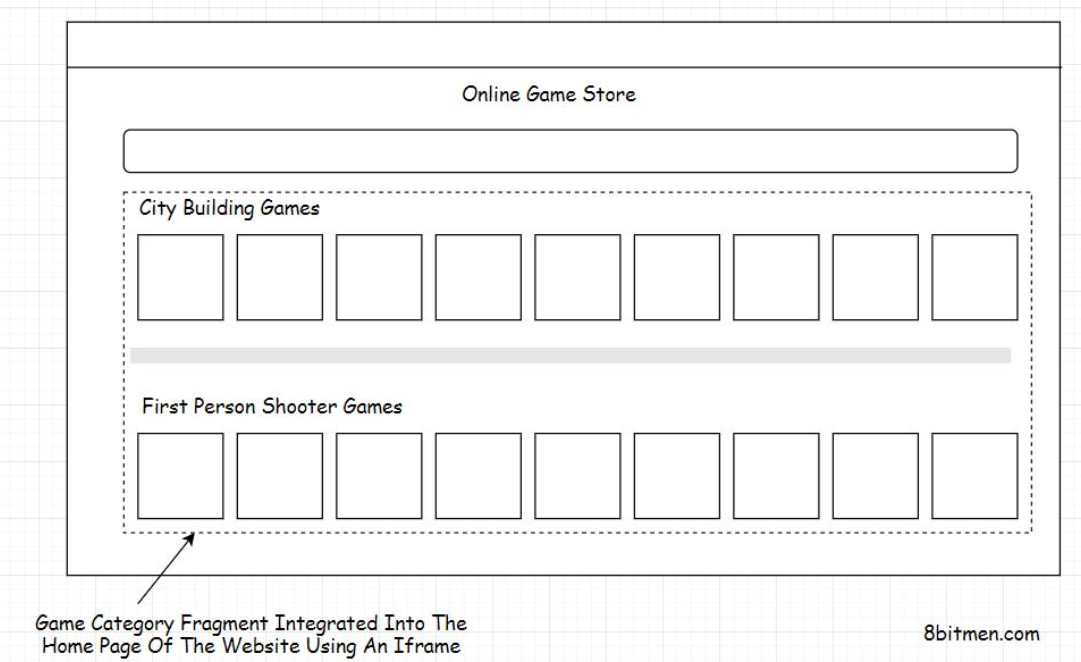
A very basic naïve way of integrating components on the client is rendering micro frontends with unique links. We just place the links on the website to enable the user to navigate to a certain micro frontend, like so:



Consider this scenario - our order checkout microservice is hosted on AWS having the URL - https://www.aws.amazon.com/onlinegamestore/checkout and our payment service is hosted on Google Cloud with the URL https://www.cloud.google.com/onlinegamestore/payment

When we integrate these micro frontends via basic links and the user navigates from the checkout page to the payment page clicking on the micro frontend link, the address in the user’s browser changes from the AWS URL to the Google Cloud URL. This will be visible to the end-user.

Also, following this basic approach, the micro frontends that need to be integrated within a specific page can be done using *Iframes*.



Well, you would have realized that this approach is not ideal. This is more like the *90s* way of building websites, connecting web pages via *direct links* and *Iframes*.

Using *IFrames* has several downsides: they can’t be bookmarked, they aren’t good from an SEO standpoint, they have stability and performance issues and so on. Besides, IFrames are an obsolete thing in the modern web dev landscape.

A recommended way to integrate components on the client-side is by leveraging a technology called the Web Components and frameworks such as [Single SPA](https://single-spa.js.org/) (<https://single-spa.js.org/> )

*Single SPA* is a *JavaScript* framework for frontend microservices that enables developers to build their frontend while leveraging different JavaScript frameworks.

If you wish to know more on web components, this [Google Chrome devs video](https://www.youtube.com/watch?v=YBwgkr_Sbx0) (<https://www.youtube.com/watch?v=YBwgkr_Sbx0> ) is a good resource.

Alright, let’s move on to understand the server-side integration process of micro frontends.

## Server side integration[#](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Server-side-integration)

When the UI components are integrated on the server, on the user request, the complete pre-built page of the website is delivered to the client from the server as opposed to sending individual micro frontends to the client and having them clubbed there.

This cuts down the website’s loading time on the client significantly since the browser does not have to do any sort of heavy lifting.

Just like the client-side integration process, we have to write additional logic on the server to integrate the micro frontends that are requested by the user.

There are several technologies and frameworks available that help us to achieve this.

## Technology and frameworks facilitating server-side integration[#](https://www.educative.io/module/lesson/web-application-architecture-101/JQOM9AMRNr9#Technology-and-frameworks-facilitating-server-side-integration)

Some of the popular frameworks that facilitate server-side integration of micro frontends are Project Mosaic, Open Components and Podium. Server Side Includes SSI is a server-side scripting language used for clubbing the content of multiple web pages on the webserver.

Zalando is a famous fashion e-commerce company in Europe that uses Project Mosaic to manage its micro frontends. Here is a talk on the micro frontends from their engineering team - [The Zalando recipe for scalable frontends](https://www.youtube.com/watch?v=m32EdvitXy4).( <https://www.youtube.com/watch?v=m32EdvitXy4> )

AutoScout24 is one of the leading internet car portals in Europe. Their engineering team leverages SSI technology to build their micro frontends. [They gave a talk](https://www.thoughtworks.com/talks/a-high-performmance-solution-to-microservice-ui-composition) (<https://www.thoughtworks.com/talks/a-high-performmance-solution-to-microservice-ui-composition> )about their micro frontends approach.

Well, this pretty much sums up our micro frontends chapter. Let’s move on to databases in the next one.