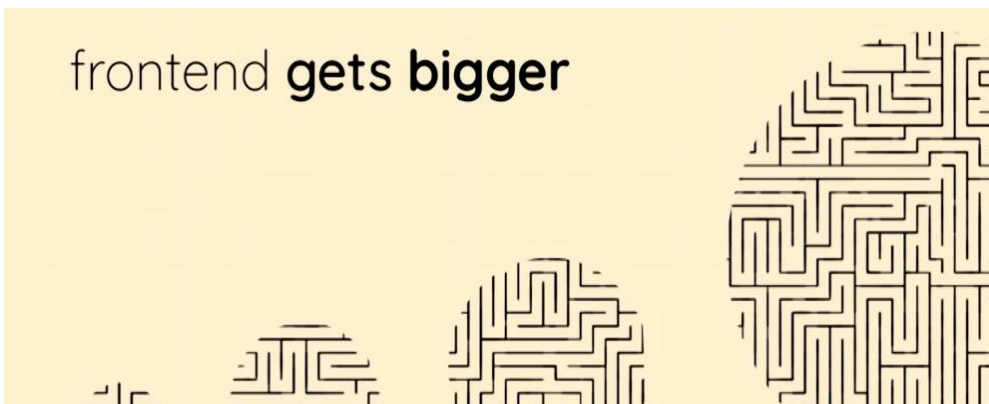



Microservices are the backend answer to handling complexity and work efficiently with multiple teams. Many frontend apps are also reaching the size where development gets harder and changing fundamental things is too risky. The software does not fit in one frontend developers head any more. Micro Frontends is a way of cutting your software into pieces that can live on their own and are each operated by an end-to-end team. It's not microservices for frontend people. Think of it as full stack microservices that come with database and user interface. This solves many issues. One of them is bridging the gap between frontend and backend developers.

Integrating the UI of multiple applications in the browser, so that the user still gets a seamless and coherent experience comes with a few challenges. This talk explains the Micro Frontends idea, when this concept is useful and what problems it tries to solve. It'll also highlight topics that you should think about when you build applications this way.



The frontend is also getting bigger



Michael Geers
Frontend Engineer

naltatis
on Twitter & GitHub

WEB REBELS



e-commerce

neuland:::
Büro für Informatik

Bremen, Germany

Micro Frontends

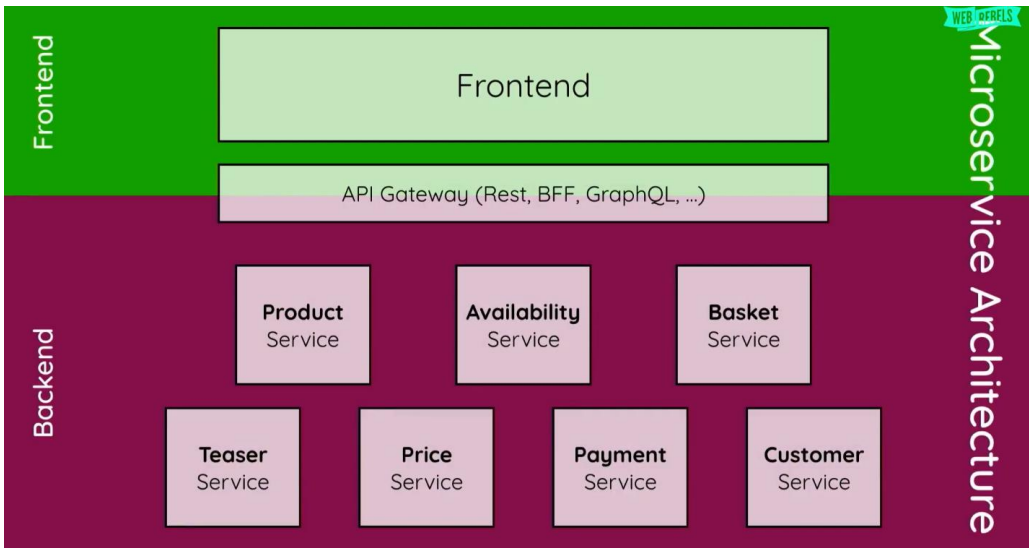
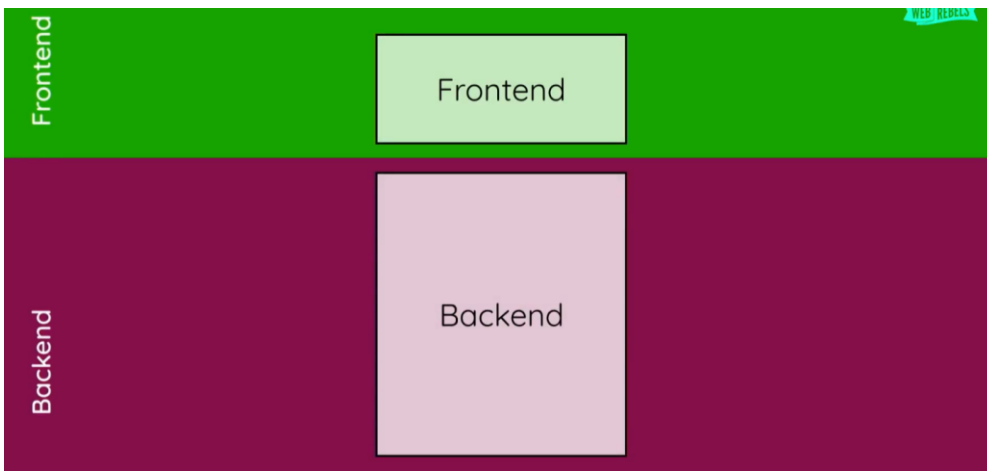
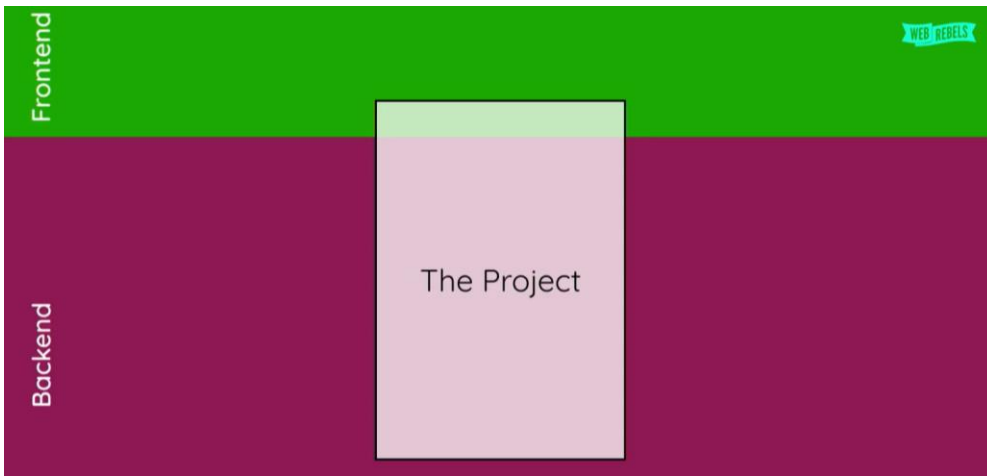
Verticalized Teams
Self Contained Systems
Vertical Decomposition
UI Composition

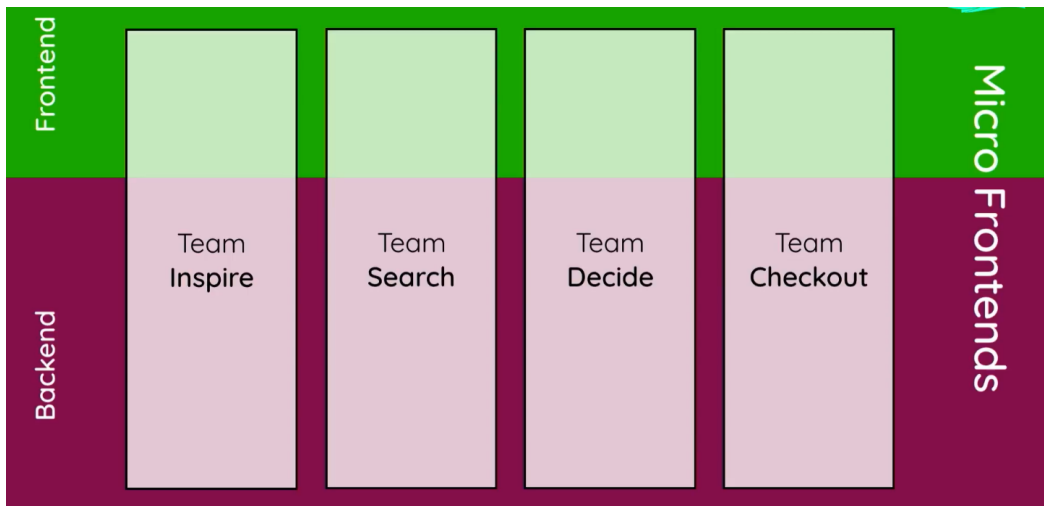
Architecture

What is it?

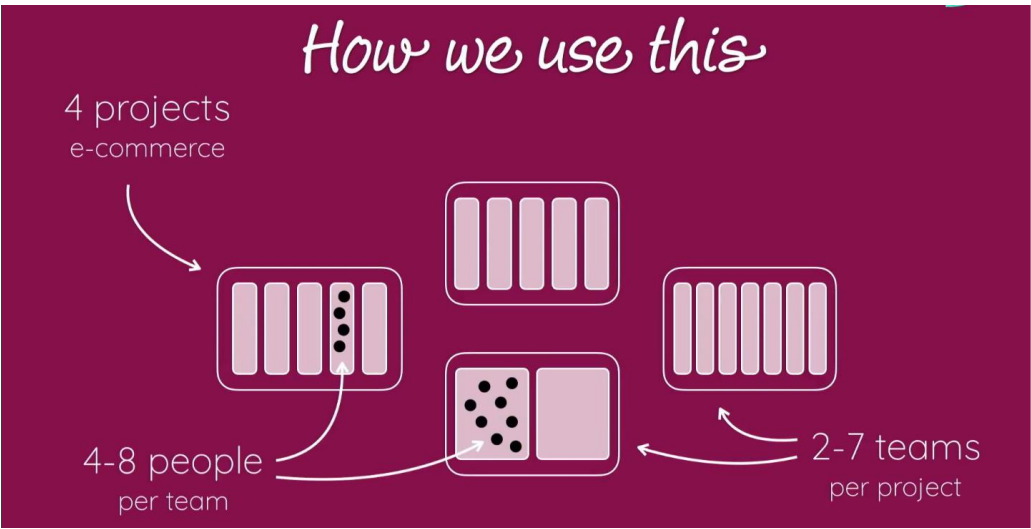
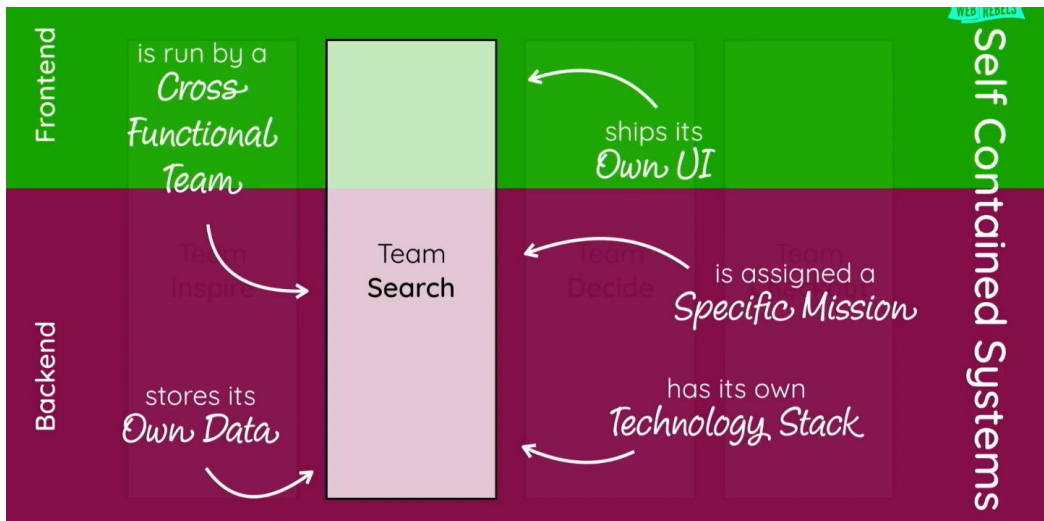
Why is it better?

How to implement?





We can instead use a micro-frontend approach and have vertical slices like the above with each team building out




Others who work like this

[Otto](#) [Ikea](#) [Facebook](#) [Amazon?](#)
[Hello Fresh](#) [Zalando](#)
[Office 365](#) [AutoScout24](#) [Spotify](#)
[FINN.no](#) [gutefrage](#)

Why Micro Frontends?



Small Autonomous Teams
with a clear mission



Customer Focus
every team ships directly
no pure api teams

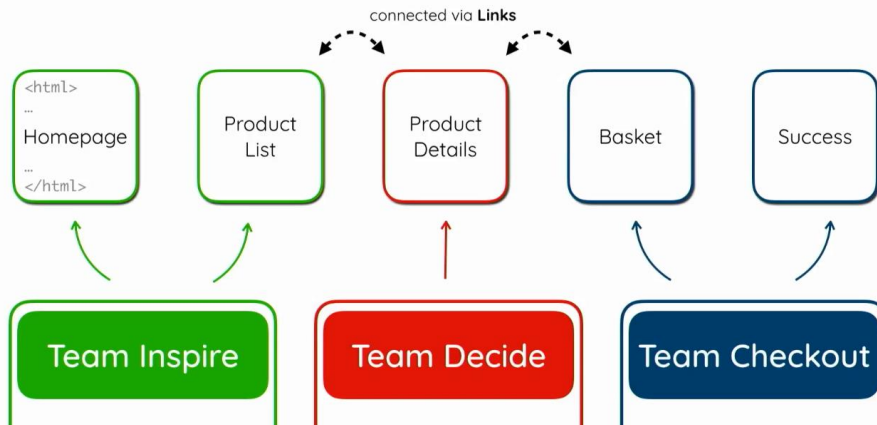


Reduced Scope
everything fits
into memory again



Frontend Renovation
without throwing everything away

A Page is Owned by one Team



The Model Store



Tractor Fendt F20 Dieselroß



buy for 54,00 €

basket: 0 item(s)

Related Products



The Model Store



Tractor Porsche-Diesel Master 419



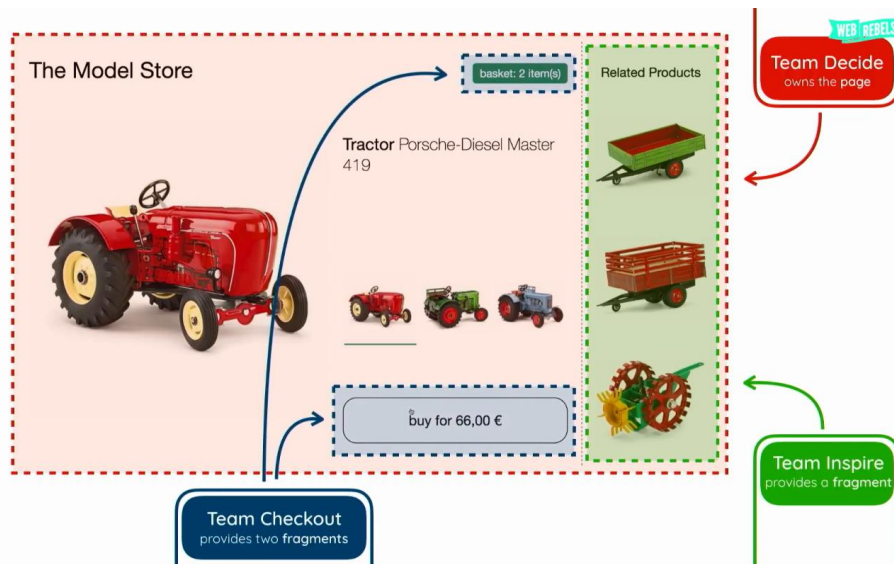
buy for 66,00 €

basket: 2 item(s)

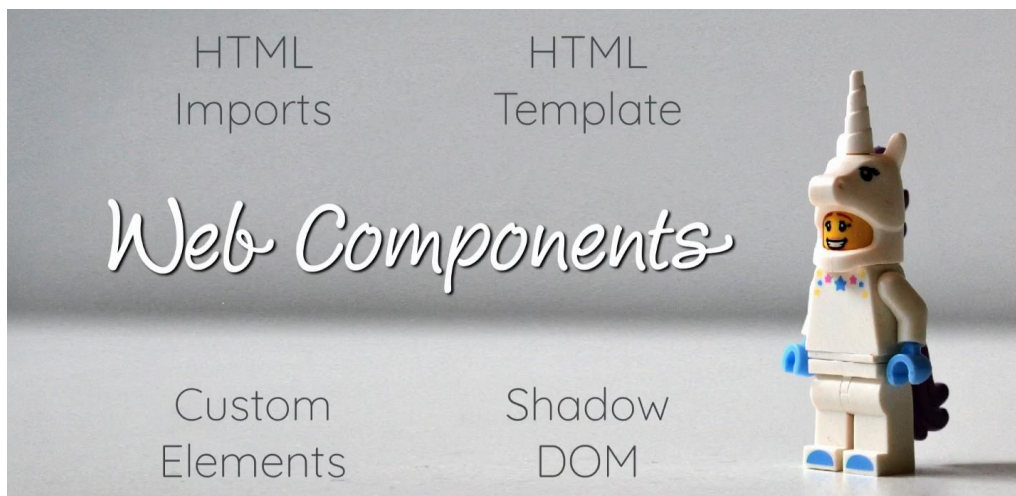
Related Products



Team Decide
owns the page



Integration in the Browser



Custom Elements

`<checkout-basket></checkout-basket>`



<https://developers.google.com/web/fundamentals/getting-started/primers/customelements>



WEB REBELS

Custom Elements

```
class CheckoutBasket extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = 'mini basket ...';  
  }  
}
```



```
customElements.define('checkout-basket', CheckoutBasket);
```

Element Lifecycle

```
class CheckoutBasket extends HTMLElement {  
  
  constructor() {...} is created  
  
  connectedCallback() {...} attached to DOM  
  
  attributeChangedCallback(attr, oldVal, newVal) {...} someone change an attribute  
  
  disconnectedCallback() {...} removed from DOM, cleanup!  
}
```

Custom Elements by Default



svelte



SkateJS

Angular
ElementshyperHTML
Element

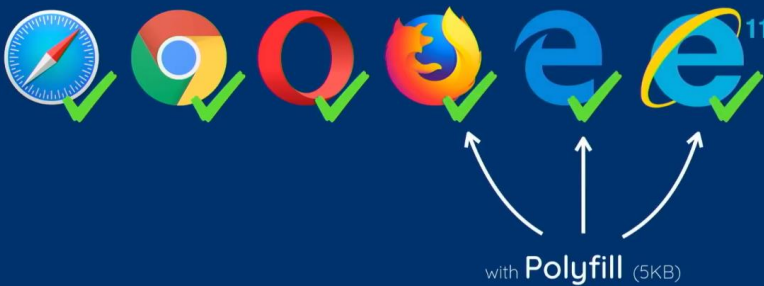
Browser Support



Custom Elements v1

API stabilized end of 2016

Browser Support



<https://github.com/WebReflection/document-register-element>

```
<checkout-basket>  
</checkout-basket>
```

basket: 0 item(s)

Tractor Porsche-Diesel Master
419



buy for 66,00 €

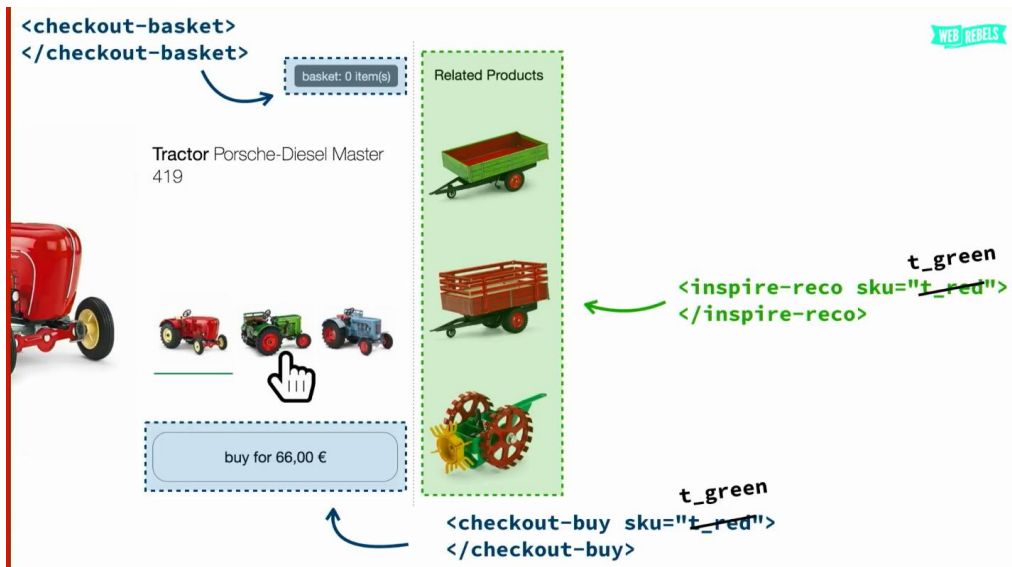
Related Products



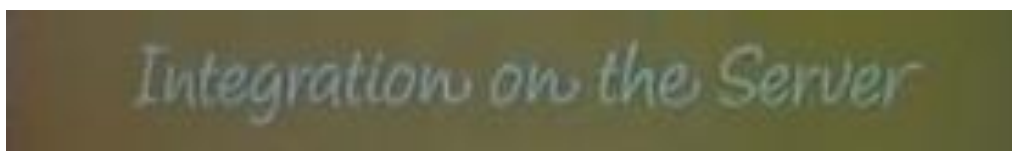
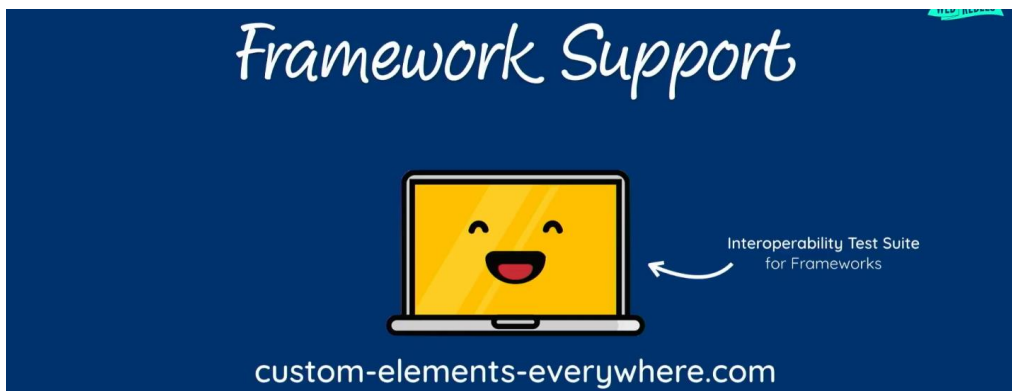
```
<inspire-reco sku="t_red">  
</inspire-reco>
```

```
<checkout-buy sku="t_red">  
</checkout-buy>
```

WEB REBELS



The red team just has to change the sku and emit an event to get the DOM to re-render again for all the affected fragments



Related Products



Server Initial Render

```
/inspire-reco?sku=t_red
```

```
res.send("<h3>Related Products</h3><ul>...")
```

Browser Rehydration & User Interaction

```
<inspire-reco sku="t_red"></inspire-reco>
```

```
el.innerHTML = "<h3>Related Products</h3><ul>...")
```

SSI Server Side Includes

```
<!--#include virtual="/some-url" -->
```

Revised for
Internet Explorer 5
and Netscape 6

JASON CRANFORD TEAGUE

DHTML AND CSS FOR THE WORLD WIDE WEB

Second Edition

Teach yourself DHTML and CSS

the quick and easy way! This

Visual QuickStart Guide uses

pictures rather than lengthy

The easiest way to import external content

into a Web page is to use a server-side

include. ... no time!



VISUAL
QUICKSTART
GUIDE

Chapter 23

Using Server-Side Includes

The easiest way to import external content into a Web page is to use a server-side include. This is not an HTML tag, but a tag that tells the computer that serves your Web pages to import external content. Although this method's success depends on whether your server understands this tag, almost all servers do these days.

To add a server-side include:

1. `<!--#include virtual="/external.html"-->`
Add the include tag anywhere in the `<body>` of your HTML document, and set it to import external.html (Code 23.3).
2. `<!--#echo var="DATE_LOCAL"-->`
Another useful server-side tag is the `<echo>` tag in which the server includes either its local time and date or Greenwich Mean Time (GMT) on the page.
3. external.html
Create a new HTML file, and save it as external.html. This file does not contain the regular `<html>` open and close tags, which are supplied by the main document—only the `<body>` tag and any HTML that could be included in a regular HTML document (Code 23.4). The results are shown in Figure 23.3.

✓ Tips

- The disadvantage of this method is that you cannot see the external content unless it is coming off a server. If you try to view this file on your local hard disk, you will see a whole lot of nothing.
- You can also add a clock that shows visitors their local time (see "Creating a Clock" in Chapter 26).

388

Code 23.3 The server-side include tags allow you to import external content, but only if the page is being delivered from a Web server.

```
<!--#include virtual="/external.html" -->
<!--#echo var="DATE_LOCAL" -->
<!--#echo var="DATE_GMT" -->
```

Code 23.4 The external content being imported into this file. This content can be any standard HTML code; just don't use `<html>` or `<body>` tags.

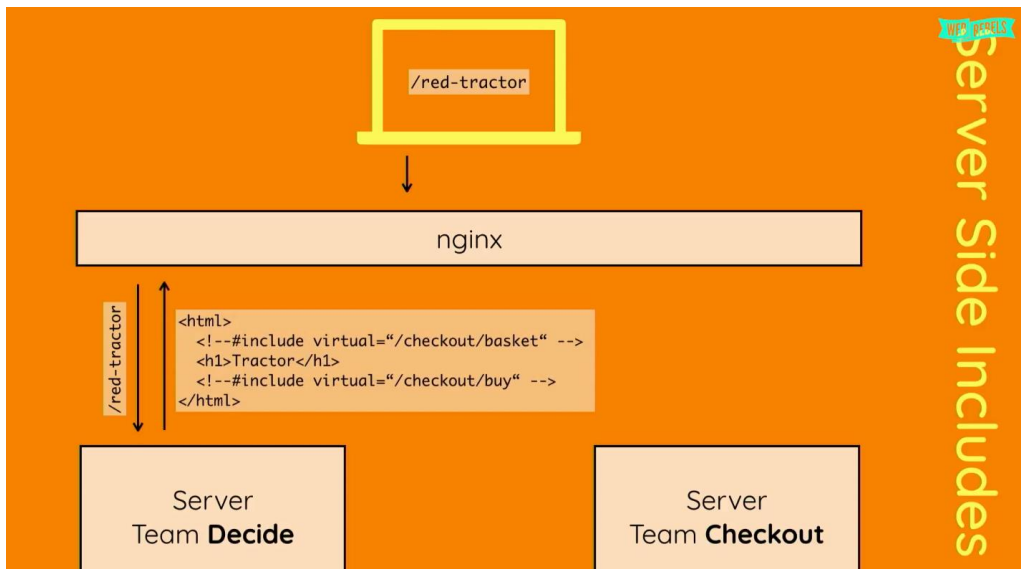
```
<div style="text-align:center">
  <h1>I'm in Wonderland!</h1>
  <h2>Chapter 1</h2>
  <p>ring a ring o' roses</p>
  <img alt="A crown" data-bbox="510 610 570 620" style="display:block; margin:auto; height:40px; width:40px;"/>
```



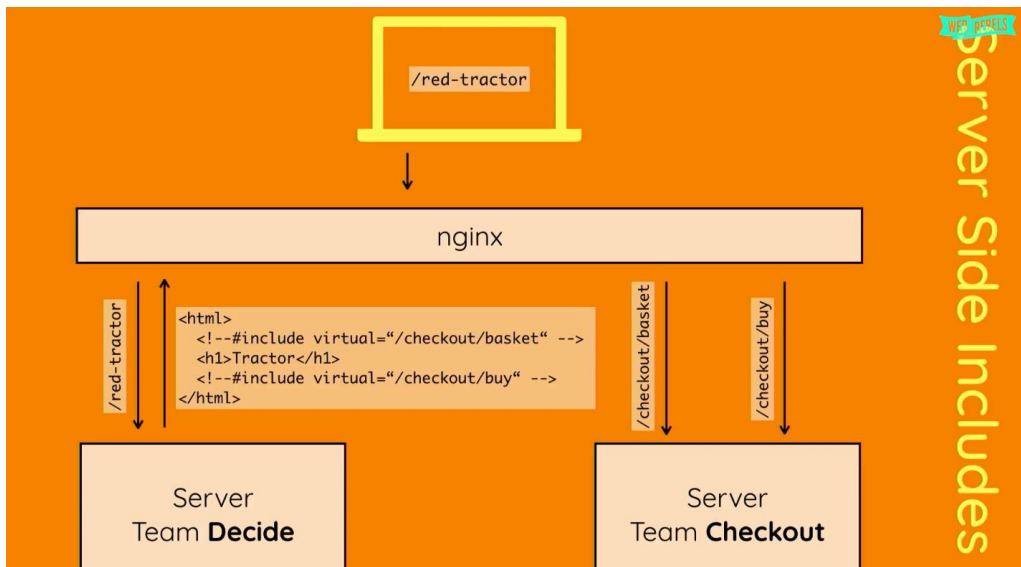
Figure 23.3 The external content has been imported successfully.

WEB REBELS

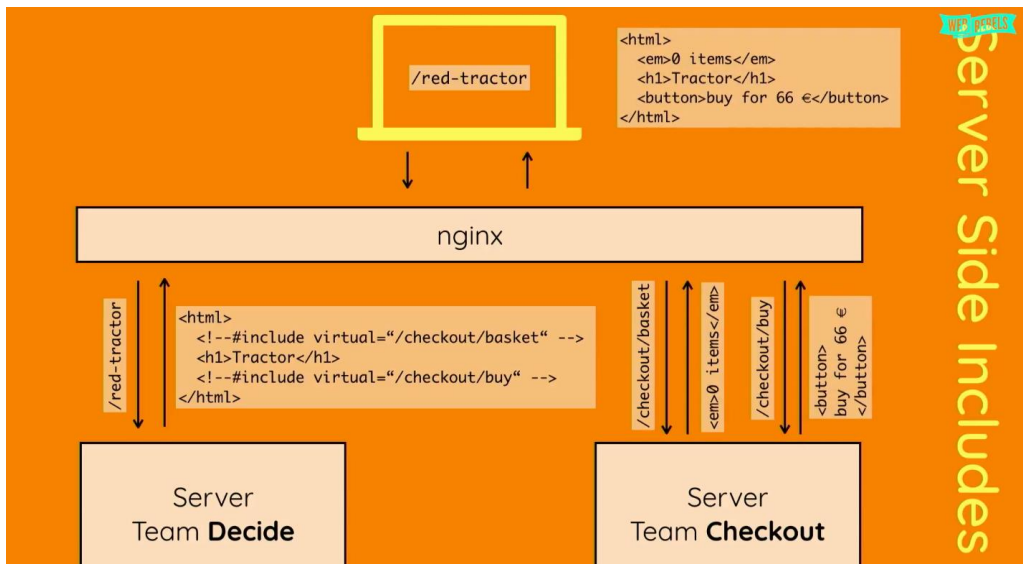
Google Books



When a request is made, the nginx web server decides which team is responsible for the request's URL and forwards the request to the correct endpoint. The complete page HTML is returned by the team server with any fragments needed as SSI tags.



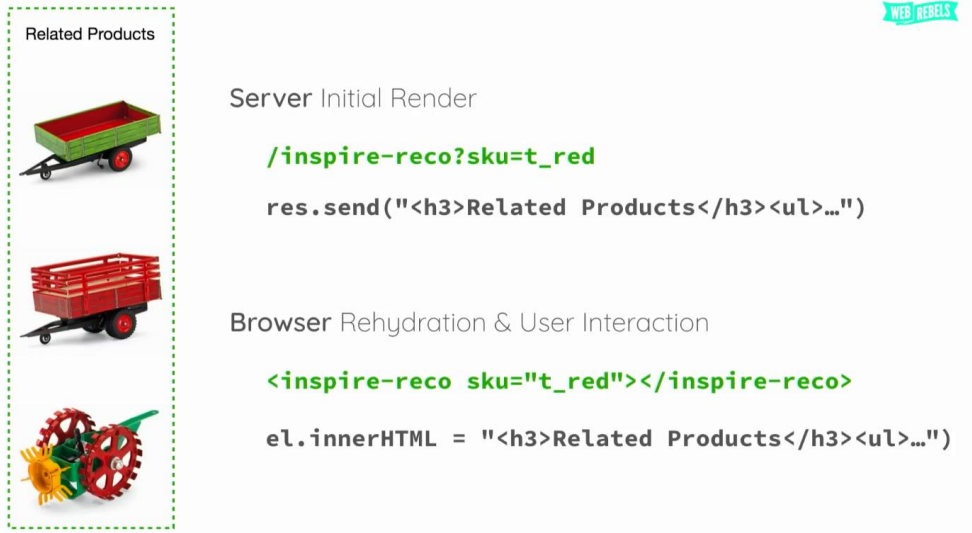
The nginx web server parses the returned HTML and makes the needed parallel calls to get the fragments from other team server locations.



The fragments are inserted into the page by the nginx web server and the complete HTML is returned to the client



Web components are just a concept that exist in the browser only, how can we benefit using the server?



It is possible to render the content of a complete fragment on a server and do the same on the client

Custom Elements ♥ SSI

client

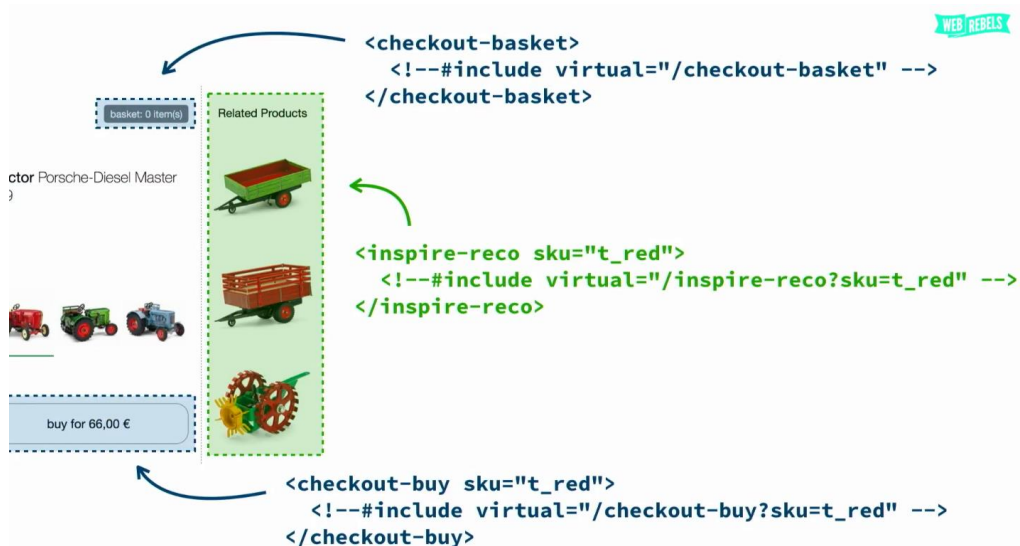
server

```
<inspire-reco sku="t_red">
```

```
<!--#include virtual="/inspire-reco?sku=t_red" -->
```

```
</inspire-reco>
```

Once the server has initially inserted the SSI fragment, the client can then take over the re-render of that fragment whenever the content changes due to an event or change within the browser



Page Transitions

Only Inside a Team

One Router per Team

Team A

Team B

/list

/product

Soft
Navigation
Client
Rendered

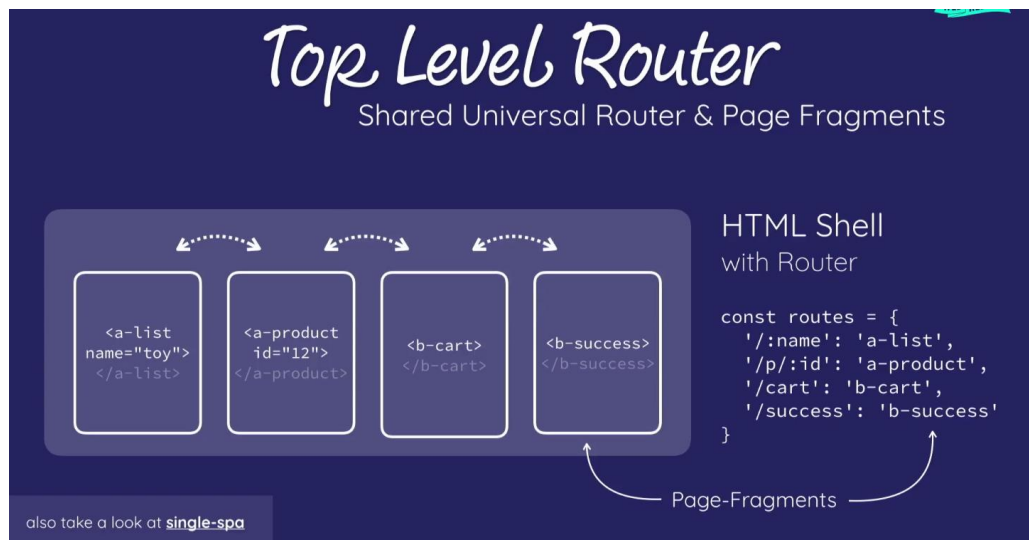
Hard
Navigation
Page
Refresh

/cart

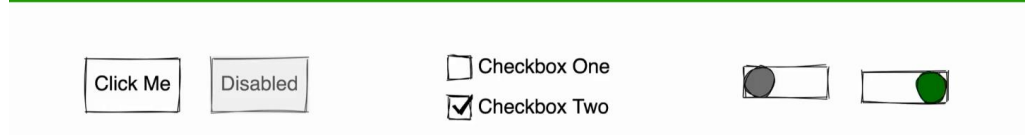
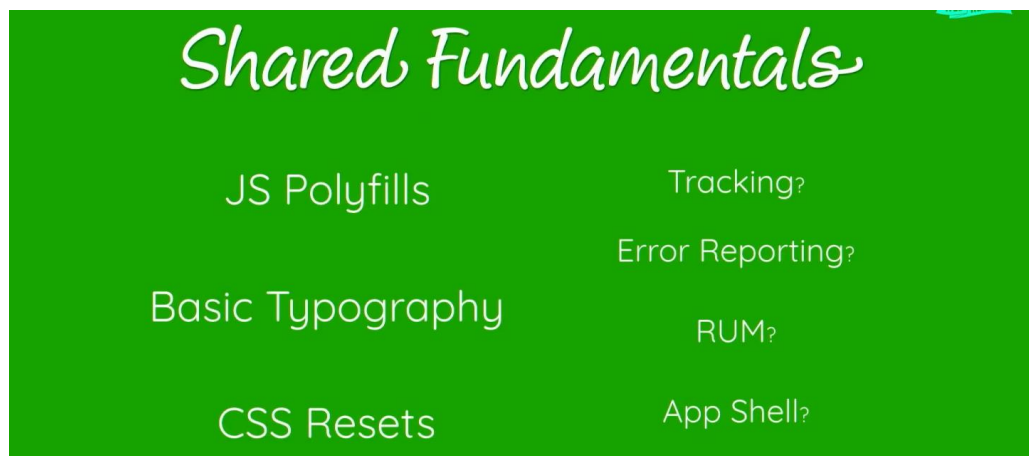
/success

Soft
Navigation
Client
Rendered

The downside with this approach is that a large number of teams will lead to more page refreshes and hard navigations that degrade user experience



In this approach, there is a HTML Shell that handles the outer page and navigation. Other teams will have to deliver fragments and routes without the header and body HTML tags to be inserted into the shell.



As Global Stylesheet

à la Bootstrap

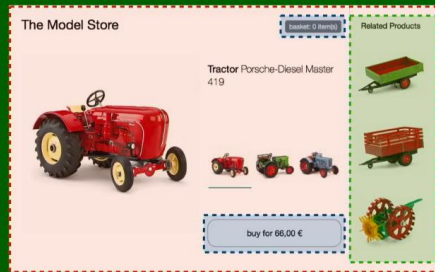
one globally included css file
includes only styling
used via css classes

Pros

- simple solution / easy to use
- consistent look

Cons

- global css tends to grow
- no easy way to detect unused css
- breaking changes are hard



cdn.example.org/global/styleguide.css

As Versioned Package

distributed via npm
components with template & styling

Pros

- only used css is shipped
- teams can upgrade independently
- fragments don't rely on outer styles

Cons

- requires build process (webpack, ...)
- common html-templating needed
- temporary visual inconsistencies



Things we've learned

Use Browser API

don't build a meta framework
avoid shared code

Isolate Teams

don't share runtime, state or globals

Pack light

register custom elements immediately
download code when needed

Talk to your Neighbors

share best practices

Ownership is important

use team prefixes when needed

Ownership is important

use team prefixes when needed

Measure Performance

HTTP/2 is great, optimize but don't overoptimize

HTTP/2 allows to load multiple JS files/bundles in parallel if needed. You don't need to include all of Angular for small app, only load library when the user needs it. Since all team code share the same runtime, it is important to use team prefixes to name event and CSS to identify when things go wrong. Try to make your design style into small libraries that teams can pull in when needed.

#perfmatters

Be reasonable!

don't use all frameworks together!
do you really need a big framework?

Measure Performance

set a performance budget & know your numbers

Don't Share State

no shared globals

Value Independence

its tempting to unify everything

Learn from Colleagues

teach them how CSS works

Talk to your Neighbors

share best practices

Ownership is important

use team prefixes when needed

Credits



Colorful Wood Structure Alexas_Fotos
<https://pixabay.com/en/background-colorful-wood-structure-3412048/>

Clocks Pixabay
<https://www.pexels.com/photo/london-new-york-tokyo-and-moscow-clocks-48770/>

Octan Tower The Lego Movie
<https://www.youtube.com/watch?v=GZT5z4K6wv4>
recaptioned by @peedlive

Action Bicycling Roman Pohorecki
<https://www.pexels.com/photo/action-bicycling-bike-biking-287598/>

Four Playing Pieces Pixabay
<https://www.pexels.com/photo/set-of-4-wooden-elongated-accessory-209651/>

Wooden Counter David Siglin
<https://www.pexels.com/photo/blur-blurry-bokeh-close-up-347139/>

The Tool Guy Tirachard Kumtanom
<https://unsplash.com/photos/UuW4psQb388>

Woodshop Igor Ovsyannykov
<https://unsplash.com/photos/XV0i4Wpdyt>

Unicorn d97jro
<https://pixabay.com/de/lego-einhorn-spielzeug-675593/>

Browser Logos Cătălin Mariș
<https://github.com/galra/browser-logos>

Broken Escalator Skitterphoto
<https://pixabay.com/photo-1746279>
<https://freesound.org/people/dobroide/sounds/15227/>
<https://freesound.org/people/sittstav/sounds/171665/>

Sad Lego Benny Cheezburger
<http://giphy.com/1BCBCKh>

Wired Elements
<https://wiredjs.com/>

Tractors Manufactum
<https://www.manufactum.com/tin-toys-c193667/>

built with Micro Frontends