

ESI Includes for Micro-FEs

Learn how to use Edge Side Includes (ESI) in your Micro-Frontend solutions.

Screenshot of the Wikipedia page for "Edge Side Includes". The page content includes:

- Section: Edge Side Includes**
From Wikipedia, the free encyclopedia
- Text:** Edge Side Includes or ESI is a small markup language for edge level dynamic web content assembly. The purpose of ESI is to tackle the problem of web infrastructure scaling.^[1] It is an application of edge computing. It is fairly common for websites to have generated content. It could be because of changing content like catalogs or forums, or because of personalization. This creates a problem for caching systems. To overcome this problem a group of companies (Akamai, Art Technology Group, BEA Systems, Circadence Corporation, Digital Island, Inc., Interwoven, Inc., Open Market, whose ESI-related technology is now owned by FatWire Software, Oracle Corporation and Vignette Corporation) developed the ESI specification and submitted it to the W3C for approval. The proposal editor was Mark Nottingham.
- Text:** ESI Language Specification 1.0 was submitted to the World Wide Web Consortium (W3C) for approval in August 2001. The W3C has acknowledged receipt, but has not accepted the proposal.^[1] ESI is implemented by some content delivery networks, such as Akamai, and by some caching proxy servers such as Varnish, Squid and Mongrel ESI,^[2] although many do not implement the complete specification.^[3] Akamai also adds additional features to the version they support.^[4]
- Section: Syntax [edit]**
ESI element tags are inserted into HTML or other text based content during creation. Instead of being displayed to viewers, these ESI tags are directives that instruct an ESI processor to take some action. The XML based ESI tags indicate to the edge-side processing agent the action that needs to be taken to complete the page's assembly. One simple example of an ESI element is the include tag which is used to include content external to the page. An ESI include tag placed in-line within an HTML document would look like:^[1]
- Text:** <esi:include src="http://example.com/1.html" alt="http://bak.example.com/2.html" onerror="continue"/>
- Text:** In this case the ESI processor would retrieve the src URL, or failing that the alt URL, or if that failed do nothing. The ESI system is usually a caching proxy server so it may have a local copy of these files which it can insert without going back to the server. Alternatively the whole page with the ESI tags may be cached, and only the ESI requests may be made to the origin server. This allows different caching times for different parts of the page, or different degrees of personalisation.
- Section: Features [edit]**
There are four main features in ESI:^[1]
 - inclusion of page fragments, as illustrated above;
 - variables which can be set from cookies or HTTP headers and then used in other ESI statements or written into markup;
 - conditions so that different markup can be used based on variables, for example if a cookie is set or not;
 - error handling, so that a failover can be used if an origin server is unavailable.
- Section: Alternatives [edit]**

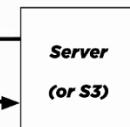
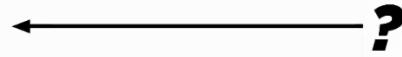
3.1 include
For example,

```
<esi:include src="http://example.com/1.html" alt="http://bak.example.com/2.html" onerror="continue"/>  
  
<esi:include src="http://example.com/ search?query=${QUERY_STRING(query)}"/>
```

The optional `src` attribute specifies an alternative resource if the `src` is not found. The requirements for the value are the same as those for `src`. Some ESI Processors may not implement this attribute, depending on its applicability; for example, surrogates near the origin server typically cannot usefully process them.

If an ESI Processor can fetch neither the `src` nor the `alt`, it returns a HTTP status code greater than 400 with an error message, unless the `onerror` attribute is present. If it is, and `onerror=continue` is specified, ESI Processors will delete the `<esi:include>` tag.

See <https://www.w3.org/TR/esi-lang> for details of the ESI Includes specification



W3C Note

<https://www.w3.org/TR/esi-lang>

ESI-Inline/1.0

to advertise their willingness to process this tag.

3.2 choose | when | otherwise

These conditional elements add the ability to perform logic based on expressions. All three must have an end tag.

```
<esi:choose>
  <esi:when test="...">
    ...
  </esi:when>
  <esi:when test="...">
    ...
  </esi:when>
  <esi:otherwise>
    ...
  </esi:otherwise>
</esi:choose>
```

Every `choose` element must contain at least one `when` element, and may optionally contain exactly one `otherwise` element. No other ESI elements or non-ESI markup can be direct children of a `choose` element. ESI processors will execute the first `when` statement whose `test` attribute evaluates truthfully, and then exit the `choose` element. If no `when` element evaluates to true, and an `otherwise` element is present, that element's content will be executed. See "ESI Expressions" for the syntax of the `test` attribute's value.

ESI elements as well as non-ESI markup can be included inside `when` or `otherwise` elements.

For example:

```
<esi:choose>
  <esi:when test="$HTTP_COOKIE(group)==Advanced">
    <esi:include src="http://www.example.com/advanced.html"/>
  </esi:when>
  <esi:when test="$HTTP_COOKIE(group)==Basic User">
    <esi:include src="http://www.example.com/basic.html"/>
  </esi:when>
  <esi:otherwise>
    <esi:include src="http://www.example.com/new_user.html"/>
  </esi:otherwise>
</esi:choose>
```

3.3 try | attempt | except

Exception handling is provided by the `try` element, which must contain exactly one instance of both an `attempt` and an `except` element (all with end tags):

```
<esi:try>
  <esi:attempt>
    ...
  </esi:attempt>
  <esi:except>
    ...
  </esi:except>
</esi:try>
```

Valid children of `try` are `attempt` and `except`; no other ESI or non-ESI markup can be a direct child. `attempt` and `except` may contain valid ESI or non-ESI markup.

ESI Processors first execute the contents of `attempt`. A failed ESI `include` statement will trigger an error and cause the ESI Processor to execute the contents of the `except` element. Statements other than `include` and `inline` will not trigger this error.

CDN - Content Distribution Network

3.3 try | attempt | except

Exception handling is provided by the `try` element, which must contain exactly one instance of both an `attempt` and an `except` element (all with end tags):

Origin - Your server

ESI Processors first execute the contents of `attempt`. A failed ESI `include` statement will trigger an error and cause the ESI Processor to execute the contents of the `except` element. Statements other than `include` and `inline` will not trigger this error.

Edge - CDN Nodes near the customer

CDN provides a caching layer for your **Origin** servers called the **Edge**. ESI's are handled by the CDN

https://jher.github.io/micro-fe-static/

The Model Store

basket: 0 item(s)

Tractor Porsche-Diesel Master 419

buy for 66,00 €

Related Products

https://github.com/jher/micro-fe-static

Search or jump to... Pull requests Issues Marketplace Explore

jher / micro-fe-static

Unwatch 1 ★ Star 0 Fork 0

Standard micro-fe example in pure HTML, CSS, JS

Manage topics

4 commits 1 branch 0 packages 0 releases 1 environment 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Description	Commit
index.html	First commit	Latest commit c0eb8cc 10 minutes ago
page.css	Update index.html	23 hours ago
images	First commit	10 minutes ago

Add a README

```

iTerm2 Shell Edit View Session Scripts Profiles Toolbar Window Help
→ tmp mkdir esi
→ tmp cd esi
→ esi npx degit git@github.com:jherr/micro-fe-static.git origin
npx: installed 32 in 2.289s
> cloned jherr/micro-fe-static#master to origin
→ esi code .

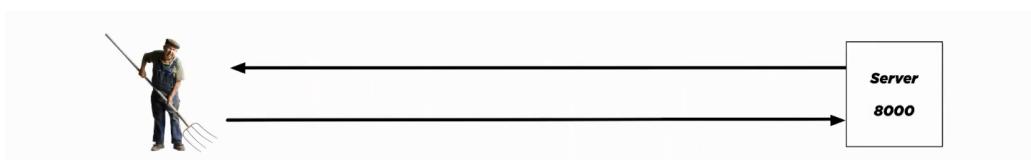
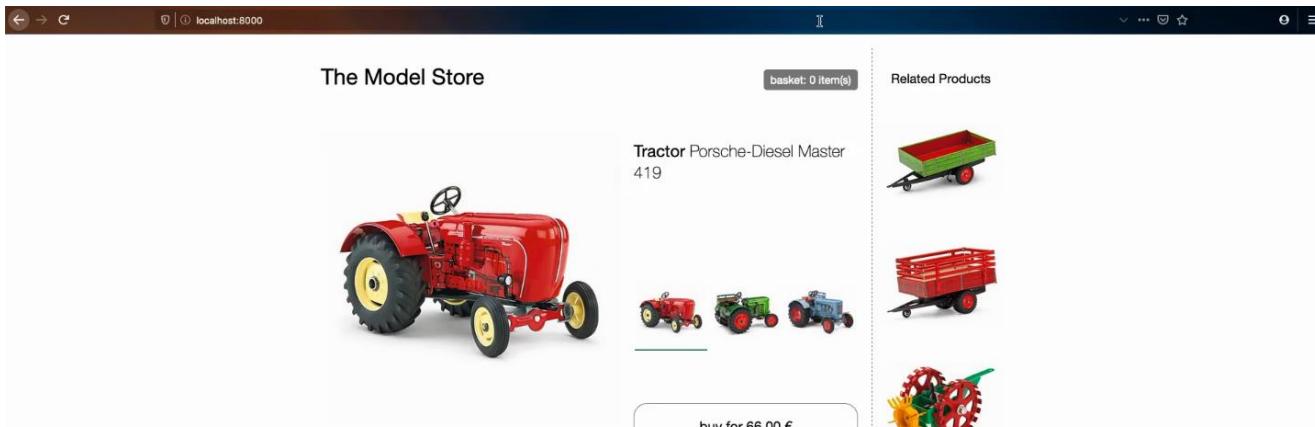
```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
→ esi cd origin
→ origin python -m SimpleHTTPServer 8000
Serving HTTP on 0.0.0.0 port 8000 ...

```



```

index.html — esi
origin > index.html > ↗ html > ↗ body > ↗ main/app
14 <div id="options">
15   <button class="active" type="button" data-sku="t_porsche">
16     
17   </button>
18
19   <button class="" type="button" data-sku="t_fendt">
20     
21   </button>
22
23   <button class="" type="button" data-sku="t_eicher">
24     
25   </button>
26 </div>
27 <button id="buy" type="button">buy for 66,00 €</button>
28 <div id="reco">
29   <h3>Related Products</h3>
30   
31   
32   
33 </div>
34 </main>
35 </body></html>

```

What we want to do is to take the **recommendations** **HTML section** out, put it in a new **HTML** pages within the components folder and put it back in the original page using an ESI tag.

```

<div id="reco">
  <h3>Related Products</h3>
  
  
  
</div>

```

```

<div id="options">
  <button class="active" type="button" data-sku="t_porsche">
    
  </button>

  <button class="" type="button" data-sku="t_fendt">
    
  </button>

  <button class="" type="button" data-sku="t_eicher">
    
  </button>
</div>
<button id="buy" type="button">buy for 66,00 €</button>
<esi:include src="http://localhost:8050/recommendations.html"></esi:include>

```

We are going to run the recommendations page out on port **8050** and then use the **ESI include** to get it from there

The Model Store

basket: 0 item(s)

Tractor Porsche-Diesel Master 419





buy for 66,00 €

```

<div id="options">
  <button class="active" type="button" data-sku="t_porsche">
    
  </button>

  <button class="" type="button" data-sku="t_fendt">
    
  </button>

  <button class="" type="button" data-sku="t_eicher">
    
  </button>
</div>
<button id="buy" type="button">buy for 66,00 €</button>
<esi:include src="http://localhost:8050/recommendations.html"></esi:include>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

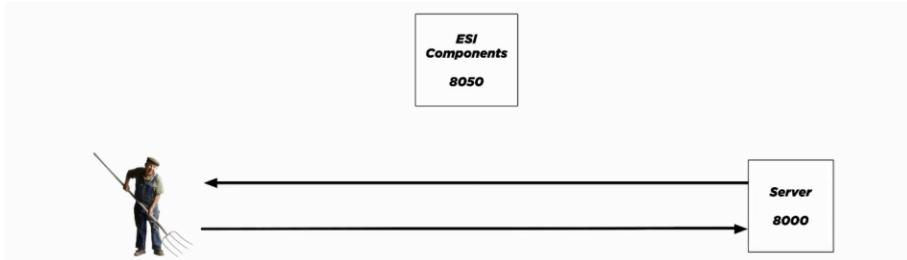
```

→ esi cd components
→ components python -m SimpleHTTPServer 8050
Serving HTTP on 0.0.0.0 port 8050 ...

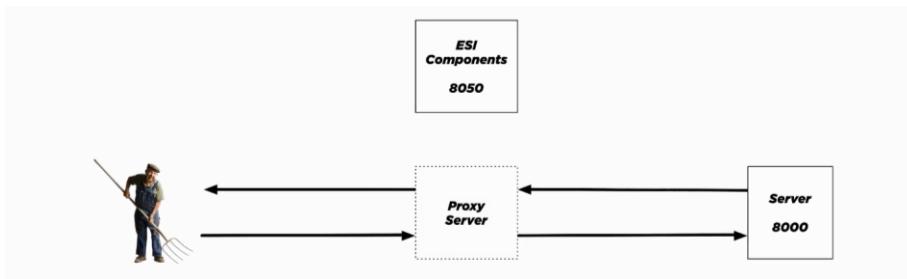
```



Note that there is no custom styling here, the host page is the one providing the styling



We need something between the 2 servers to resolve the ESI Include tags for us



The CDN will be acting as our proxy server that helps to resolve the ESI include tags before serving the host page content

The screenshot shows a browser-based code editor interface. The top navigation bar includes 'Code', 'File', 'Edit', 'Selection', 'View', 'Go', 'Debug', 'Terminal', 'Window', and 'Help'. The left sidebar has sections for 'EXPLORER', 'OPEN EDITORS', and 'ESI'. Under 'OPEN EDITORS', there are two tabs: 'index.html' (active) and 'recommendations.html'. Under 'ESI', there's a 'components' folder containing 'recommendations.html'. The main content area displays the 'index.html' code:

```
<div id="options">
  <button class="active" type="button" data-sku="t_porsche">
    
  </button>

  <button class="" type="button" data-sku="t_fendt">
    
  </button>
```

We will create a proxy folder where we will create a NodeJS Express Proxy Server

```

index.html — esi
origin > index.html > ...
14   <div id="options">
15     <button class="active" type="button" data-sku="t_porsche">
16       
17     </button>
18
19     <button class="" type="button" data-sku="t_fendt">
20       
21     </button>
22
23     <button class="" type="button" data-sku="t_eicher">
24       
25     </button>
26   </div>
27   <button id="buy" type="button">buy for 66,00 €</button>
28   <esi:include src="http://localhost:8050/recommendations.html"/></esi:include>
29 </main>
30 </body></html>
31
warning The yes flag has been set. This will automatically answer yes to all questions, which may have security implications
.
success Saved package.json
+ Done in 0.03s.
→ proxy yarn add express express-http-proxy
yarn add v1.19.2
info No lockfile found.
[1/4] ⚡ Resolving packages...
· unpipe@1.0.0

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
.
success Saved package.json
+ Done in 0.03s.
→ proxy yarn add express express-http-proxy
yarn add v1.19.2
info No lockfile found.
[1/4] ⚡ Resolving packages...
[2/4] 🚀 Fetching packages...
[########################################] 55/57

```

Introducing npm Teams! Private packages, team management tools. [Learn more »](#)

express-http-proxy

1.6.0 · Public · Published 3 months ago

- [Readme](#)
- [Explore](#)
- [3 Dependencies](#)
- [342 Dependents](#)
- [37 Versions](#)

express-http-proxy npm package 1.6.0 build passing

Express middleware to proxy request to another host and pass response back to original caller.

Install

```
$ npm install express-http-proxy --save
```

Usage

```
proxy(host, options);
```

Example:
To proxy URLs starting with '/proxy' to the host 'www.google.com':

```
var proxy = require('express-http-proxy');
var app = require('express')();

app.use('/proxy', proxy('www.google.com'));
```

Install

```
> npm i express-http-proxy
```

Weekly Downloads
109,929

Version	License
1.6.0	MIT

Unpacked Size	Total Files
113 kB	57

Issues	Pull Requests
89	14

Homepage github.com/villadura/express-http-proxy

Repository github.com/villadura/express-http-proxy

We can use the **express-http-proxy** module to run a proxy server that can help resolve the ESI Include tags in our HTML

https://www.npmjs.com/package/express-http-proxy

```
$ npm install express-http-proxy --save
```

Usage

```
proxy(host, options);
```

Example:
To proxy URLs starting with '/proxy' to the host 'www.google.com':

```
var proxy = require('express-http-proxy');
var app = require('express')();

app.use('/proxy', proxy('www.google.com'));
```

Streaming
Proxy requests and user responses are piped/streamed/chunked by default.

If you define a response modifier (userResDecorator, userResHeaderDecorator), or need to inspect the response before continuing (maybeSkipToNext), streaming is disabled, and the request and response are buffered. This can cause performance issues with large payloads.

Promises
Many function hooks support Promises. If any Promise is rejected, next(x) is called in the hosting application, where x is whatever you pass to `Promise.reject`;

e.g.

```
app.use(proxy('/reject-promise', {
  proxyReqOptDecorator: function() {
    return Promise.reject('An arbitrary rejection message.');
  }
}));
```

Last publish
3 months ago

Collaborators

Homepage [github.com/villadora/express-http-proxy...](#)

Repository [github.com/villadora/express-http-proxy](#)

Try on RunKit

Report a vulnerability

https://www.npmjs.com/package/express-http-proxy

```
resolve(req.method === 'GET');
});
}
});
```

Note that in the previous example, `resolve(false)` will execute the happy path for filter here (skipping the rest of the proxy, and calling `next()`). `reject()` will also skip the rest of proxy and call `next()`.

userResDecorator (was: intercept) (supports Promise)

You can modify the proxy's response before sending it to the client.

```
app.use('/proxy', proxy('www.google.com', {
  userResDecorator: function(proxyRes, proxyResData, userReq, userRes) {
    data = JSON.parse(proxyResData.toString('utf8'));
    data.newProperty = 'exciting data';
    return JSON.stringify(data);
  }
}));
```

```
app.use(proxy('httpbin.org', {
  userResDecorator: function(proxyRes, proxyResData) {
    return new Promise(function(resolve) {
      proxyResData.funkyMessage = 'oi io oo ii';
      setTimeout(function() {
        resolve(proxyResData);
      }, 200);
    });
  }
}));
```

304 - Not Modified
When your proxied service returns 304, not modified, this step will be skipped, since there is no body to decorate.

We can use the use decorator function to resolve the ESI Include tags within our proxy server before passing it on

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays the 'OPEN EDITORS' section with files like 'index.html', 'recommendations.html', and 'proxy'. Below this is the 'ESI' folder containing 'components', 'origin', 'images', 'index.html', and '# page.css'. The main editor area shows the 'index.html' file content:

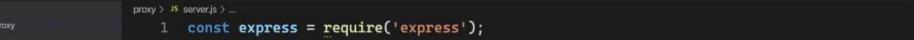
```
<div id="options">
  <button class="active" type="button" data-sku="t_porsche">
    
  </button>

  <button class="" type="button" data-sku="t_fendt">
    
  </button>

  <button class="" type="button" data-sku="t_eicher">
    
  </button>
</div>
<button id="buy" type="button">buy for 66,00 €</button>
<esi:include src="http://localhost:8050/recommendations.html"></esi:include>
</main>
</body></html>
```

The bottom of the editor shows tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, displaying the following command history:

```
raw-body@2.4.1
safer-buffer@2.1.2
serve-static@1.14.1
type-is@1.6.18
utils-merge@1.0.1
vary@1.1.2
Done in 1.06s.
proxy touch server.js
proxy
```

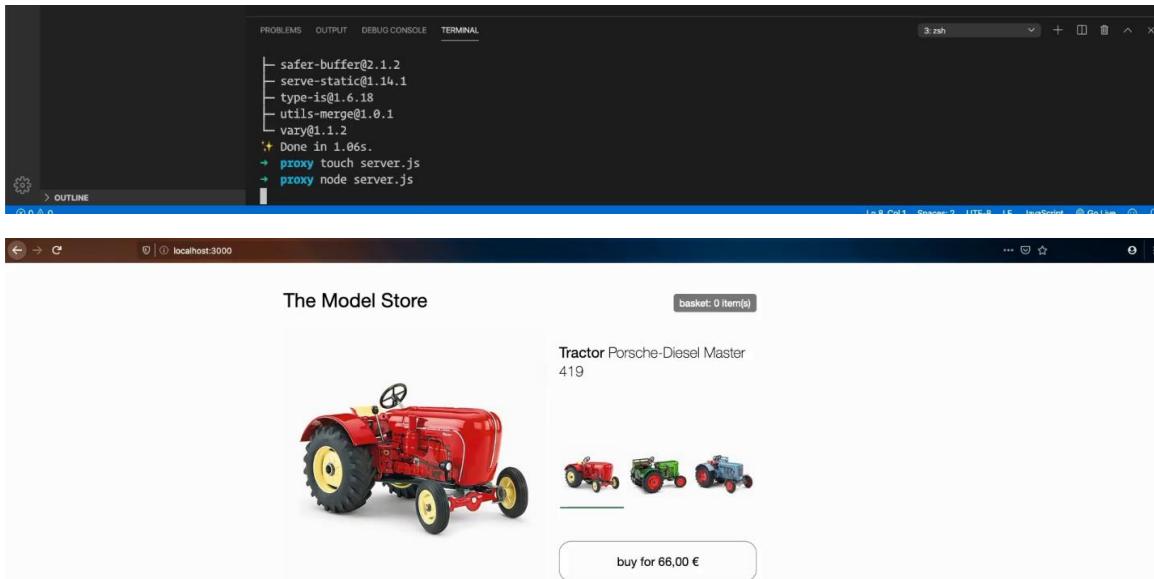


```
const express = require('express');
const proxy = require('express-http-proxy');

const app = express();
app.use('/', proxy('http://localhost:8000', {}));

app.listen(3000);
```

We then create our `server.js` file and run the proxy server out of port 8000. We will initially let everything pass through.



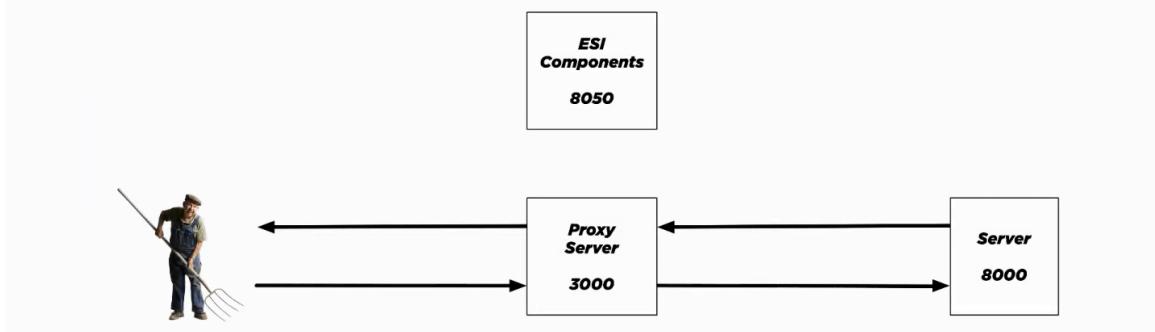
The host page looks the same as before, the recommendations HTML code is not being resolved or displayed

```

<html lang="en">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta href="style.css" rel="stylesheet">
</head>
<body>
<div id="app">
<h1 id="store">The Model Store</h1>
<div class="empty" data-sku="empty">Basket: 0 item(s)</div>
<div id="image"><div></div></div>
<div id="name">Tractor <small>Plain JavaScript, One Texas, Client Render</small></div>
<div id="location">Austin, TX</div>
<button class="active" type="button" data-sku="t_porsche">
 Porsche Diesel Master 419</button>
<button type="button" data-sku="t_feudt">
 Fendt F20 Dieselrodt</button>
<button class="" type="button" data-sku="t_eicher">
 Eicher Diesel 215/16</button>
</div>
<button id="buy" type="button">Buy for 66,00</button>
<esi:include src="http://localhost:8050/recommendations.html"></esi:include>
</div>
</body>

```

We now have the ESI Include tag passing through unresolved



We need to put the code to resolve the ESI Include tags for the recommendations HTML code

```

OPEN EDITORS
proxy > JS server.js > userResDecorator
1 const express = require('express');
2 const proxy = require('express-http-proxy');
3
4 const app = express();
5 app.use('/', proxy('http://localhost:8000'), {
6   userResDecorator: function(proxyRes, proxyResData) {
7     return proxyResData;
8   }
9 });
10
11 app.listen(3000);
12

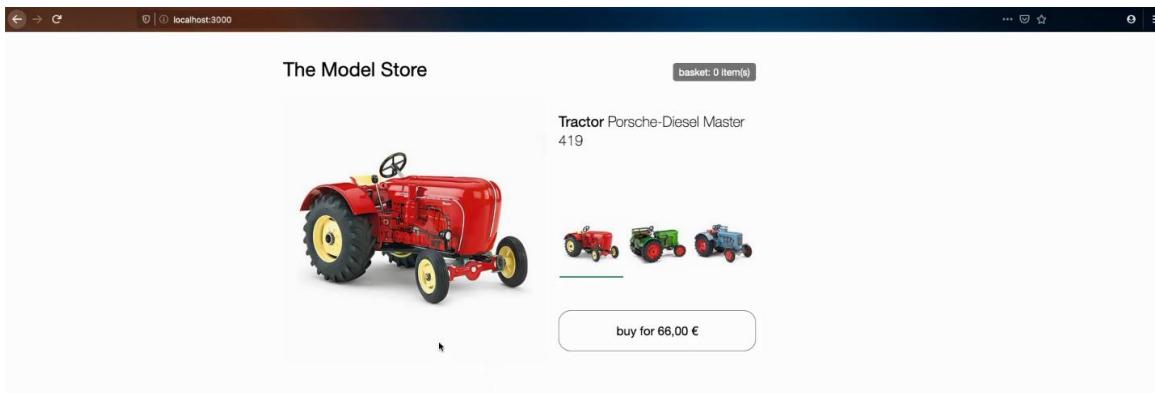
```

```

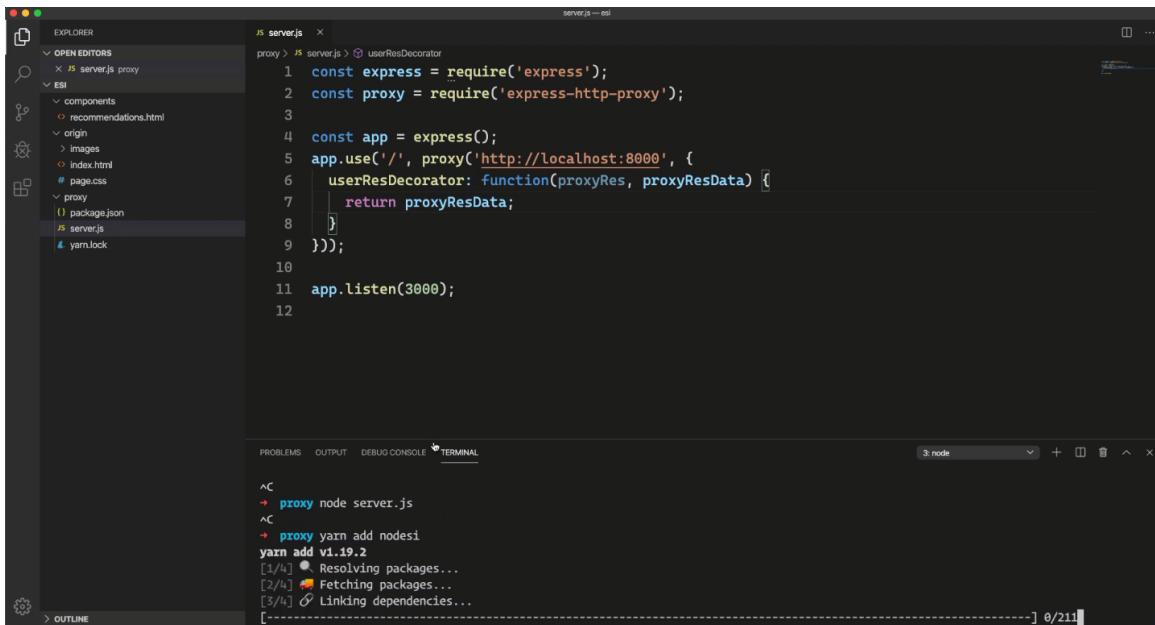
serve-static@1.14.1
type-is@1.6.18
utils-merge@1.0.1
vary@1.1.2
  Done in 1.06s.
  proxy touch server.js
  proxy node server.js
  proxy

```

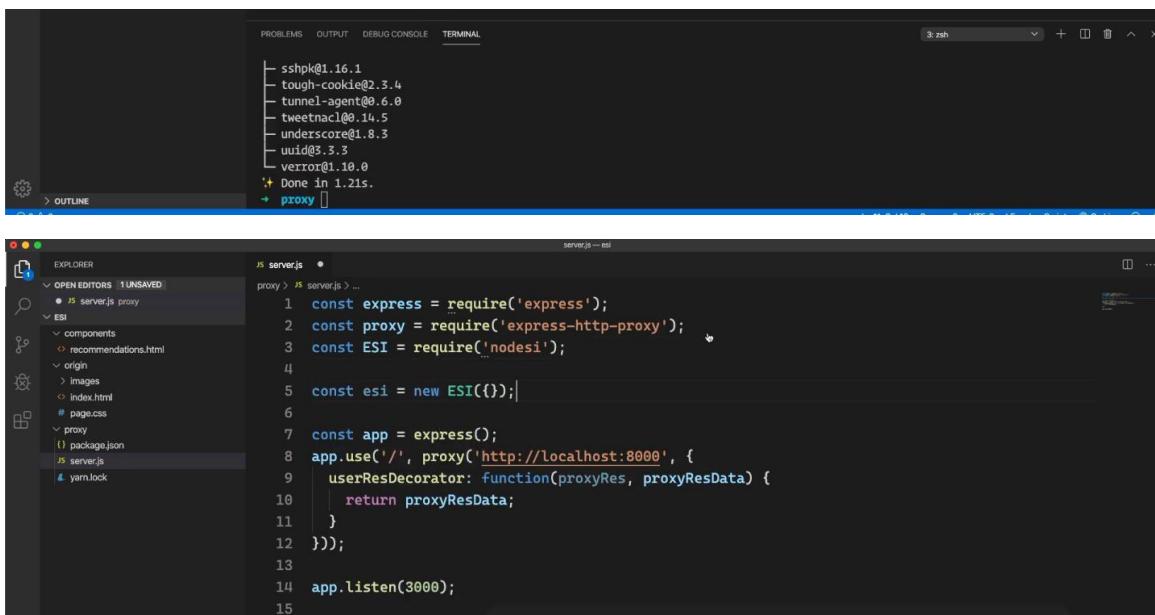
We can now set up the *user response decorator function* `userResDecorator()` discussed in the NPM page for the **express-http-proxy** and just pass things through for now.



Things still work as before.



We then add the **nodesi** package using **yarn add nodesi**, the module that does ESI parsing and also handles those requests for us.



With base URL for relative paths:

```
...  
app.use(esimiddleware());  
  
app.get('/example', function(req, res) {  
  req.esiOptions = {  
    baseUrl: req.url  
  };  
  res.render('example');  
});
```

With headers:

```
var ESI = require('nodesi');  
  
var esi = new ESI({  
  baseUrl: 'http://full-resource-path'  
});  
esi.process('<esi:include src="/stuff.html" />').then(function(result)  
  // result is a fetched html  
);
```

server.js — esi

EXPLORER

- OPEN EDITORS
 - JS server.js
 - JS server.js proxy
- ESI
 - components
 - recommendations.html
 - origin
 - images
 - index.html
 - page.css
 - proxy
 - package.json
 - server.js
- yarn.lock

server.js

```
1 const express = require('express');
2 const proxy = require('express-http-proxy');
3 const ESI = require('nodesi');
4
5 const esi = new ESI();
6
7 const app = express();
8 app.use('/', proxy('http://localhost:8000'), {
9   userResDecorator: function(proxyRes, proxyResData) {
10     return proxyRes.headers['content-type'] === 'text/html' ?
11       esi.process(proxyResData.toString()) : proxyResData;
12   }
13 });
14
15 app.listen(3000);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

sshpk@1.16.1
 - tough-cookie@2.3.4
 - tunnel-agent@0.6.0
 - tweetnacl@0.14.5
 - underscore@1.8.3
 - uuid@3.3.3
 - verror@1.10.0
 ✨ Done in 1.21s.
 ➜ proxy

The Model Store

basket: 0 item(s)

Related Products

Tractor Porsche-Diesel Master 419



[buy for 66,00 €](#)



Now we have the ESI Include tag for the recommendations HTML resolved correctly and we see the recommendations

```

<html lang="en"><head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta href="style.css" rel="stylesheet">
</head>
<body>
<div id="app">
<h1 id="store">>The Model Store</h1>
<div id="basket">>Basket</div>
<div id="name">>Tractor <small>Porsche-Diesel Master 419</small></div>
<div id="options">>Options</div>
<div class="active" type="button" data-sku="t_porsche">

</button>
<div class="active" type="button" data-sku="t_fendt">

</button>
<div class="active" type="button" data-sku="t_eicher">

</button>
<div id="buy">>Buy</div>
<div id="products">>Products</div>



</div>
</div>
</main>
</body></html>

```

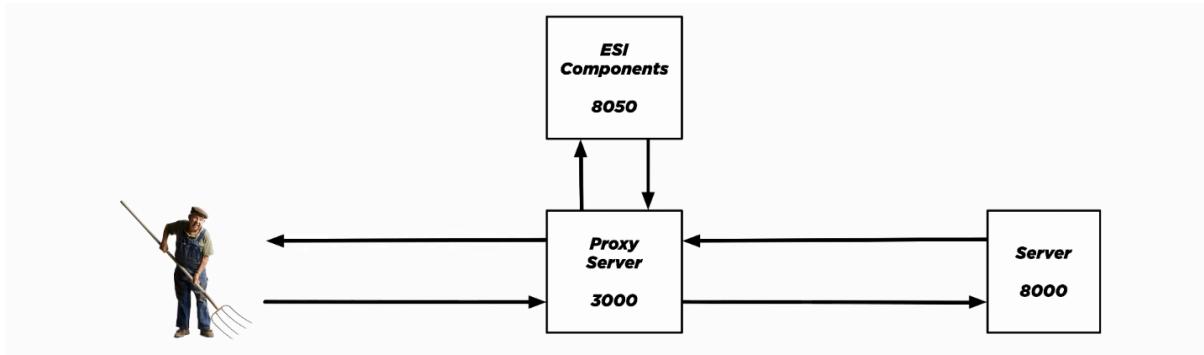
Screenshot of a developer's environment showing the code for `recommendations.html`. The code contains a section for related products with three images labeled Reco 3, Reco 5, and Reco 6.

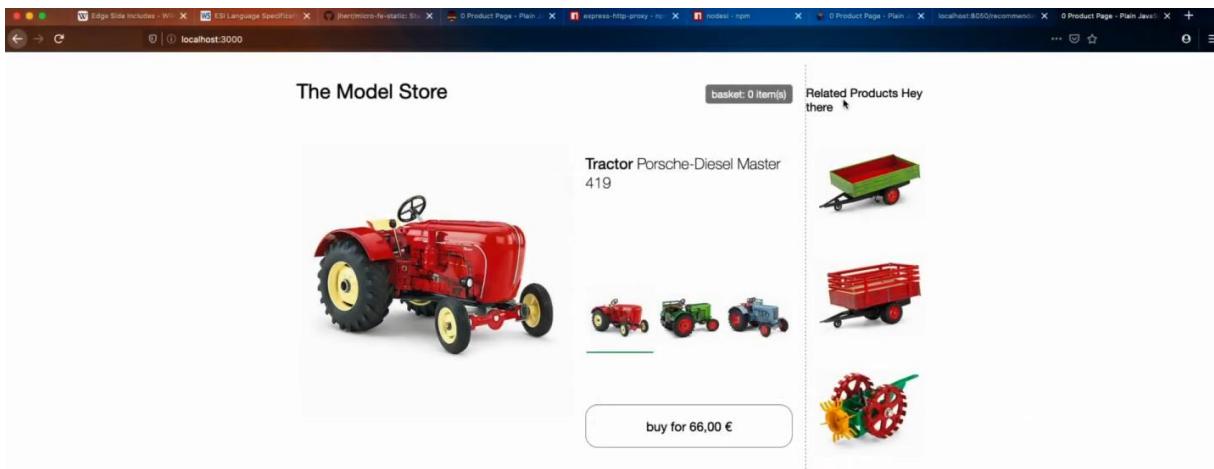
```

Code File Edit Selection View Go Debug Terminal Window Help
recommendations.html — es6
EXPLORER JS server.js components > recommendations.html components
components > recommendations.html > div#reco > h3
1  <div id="reco">
2    <h3>Related Products Hey there!</h3>
3    
4    
5    
6  </div>
7

```

Screenshot of a browser window showing the product page for a Porsche-Diesel Master 419 tractor. The page includes a basket summary, a large image of the tractor, a price of 66,00 €, and a sidebar with related products.





Let us now show how to make the code in the original page interact with those injected ESI components

```

Code File Edit Selection View Go Debug Terminal Window Help
EXPLORE server.js recommendations.html index.html — esi
origin > index.html > <html> <body> <main>
17   <button class="" type="button" data-sku="t_fendt">
18     
19   </button>
20
21   <button class="" type="button" data-sku="t_eicher">
22     
23   </button>
24 </div>
25   <button id="buy" type="button">buy for 66,00 </button>
26   <esi:include src="http://localhost:8050/recommendations.html"></esi:include>
27 </main>
28   <script src="page.js"></script>
29 </body></html>
30
31
32

```

```

Code File Edit Selection View Go Debug Terminal Window Help
EXPLORE server.js recommendations.html index.html — page.js
origin > JS page.js > ...
1  let cartCount = 0;
2  document.getElementById('buy').addEventListener('click', () => {
3    cartCount += 1;
4    window.dispatchEvent(new CustomEvent('updateCart', {
5      detail: {
6        cartCount,
7      }
8    }));
9  })

```

We start by creating some code for the buy button interactivity on the page in the **page.js** file. This code will add 1 to the **cartCount** value and put it into a **custom event on the window** saying that there is a new cart value.

The Model Store

basket: 0 item(s)

Tractor Porsche-Diesel Master 419

buy for 66,00€

Related Products Hey there

Red Tractor

Red Wagon

Green Wagon

Blue Wagon

Yellow Gears

index.html - edit

server.js proxy recommendations.html components index.html origin page.js origin ESI components recommendations.html origin images index.html page.css page.js proxy package.json server.js yarn.lock

```

    </head>
    <body>
      <main id="app">
        <h1 id="store">The Model Store</h1>
        <div id="basket" class="empty">basket: 0 item(s)</div>
        <div id="image"><div></div></div>
        <h2 id="name">Tractor <small>Porsche-Diesel Master 419</small></h2>
        <div id="options">
          <button class="active" type="button" data-sku="t_porsche">
            
          </button>
          <button class="" type="button" data-sku="t_fendt">
            
          </button>
          <button class="" type="button" data-sku="t_eicher">
            
          </button>
        </div>
      </main>
    </body>
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

+ proxy node server.js

We then extract the basket HTML code into a separate basket.html file within the component folder as below

basket.html - edit

components > basket.html ...

```

1  <div id="basket" class="empty">basket: 0 item(s)</div>
2

```

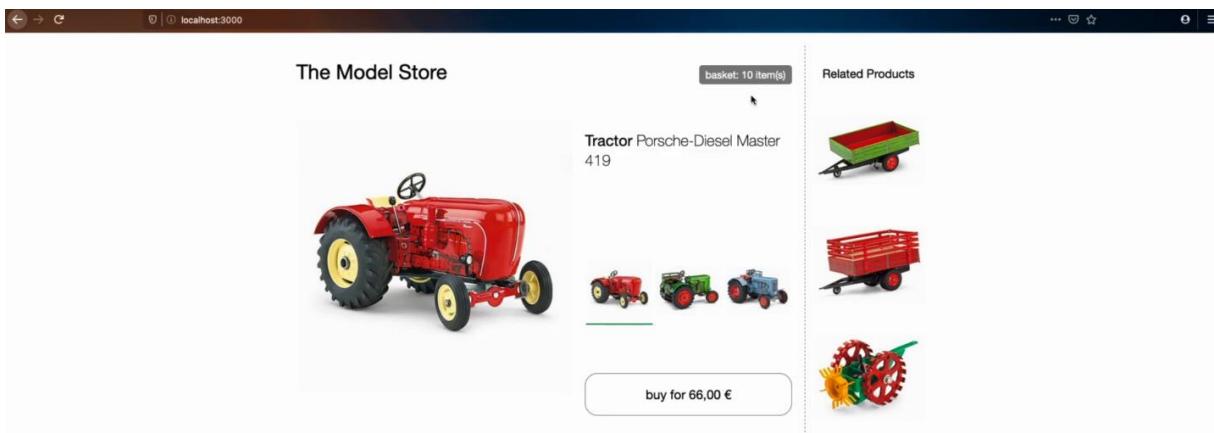
server.js proxy recommendations.html components index.html origin page.js origin ESI components basket.html recommendations.html origin images index.html page.css page.js proxy package.json server.js yarn.lock

The screenshot shows the VS Code interface with the server.js file open in the editor. The code includes ESI (Edge Side Inclusion) directives to fetch content from another origin (localhost:8050). The page displays a red tractor named 'Porsche-Diesel Master 419' with a price of 66,00 €.

```

4   <title>0 Product Page - Plain JavaScript, One Team, Client Render</title>
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <link href="page.css" rel="stylesheet">
7 </head>
8 <body>
9   <main id="app">
10    <h1 id="store">The Model Store</h1>
11    <esi:include src="http://localhost:8050/basket.html"></esi:include>
12    <div id="image"><div>
13    <h2 id="name">Tractor <small>Porsche-Diesel Master 419</small></h2>
14    <div id="options">
15      <button class="active" type="button" data-sku="t_porsche">
16        
17      </button>
18
19      <button class="" type="button" data-sku="t_fendt">
20        

```



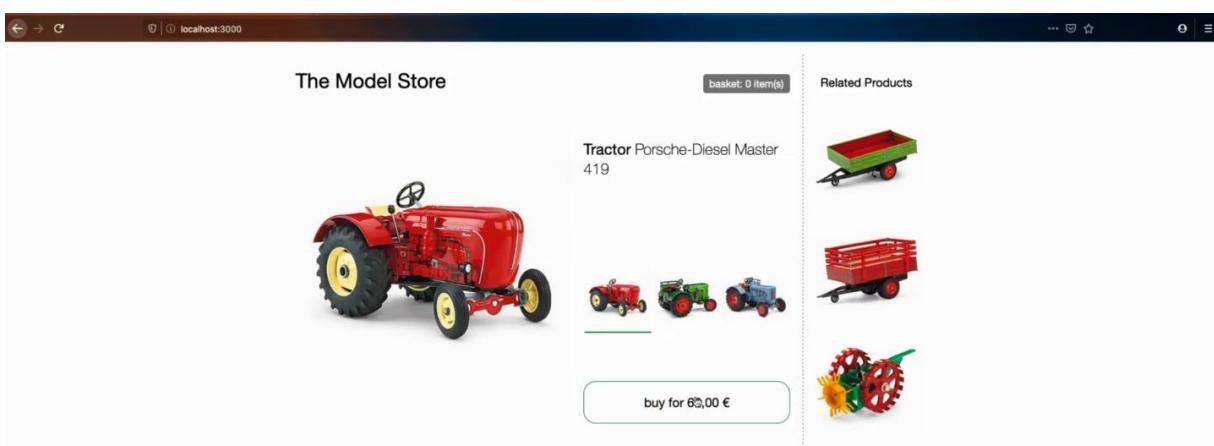
The screenshot shows the basket.html file in the editor, containing a script that listens for an 'updateCart' event and updates the 'basket' element's innerText to reflect the new cart count.

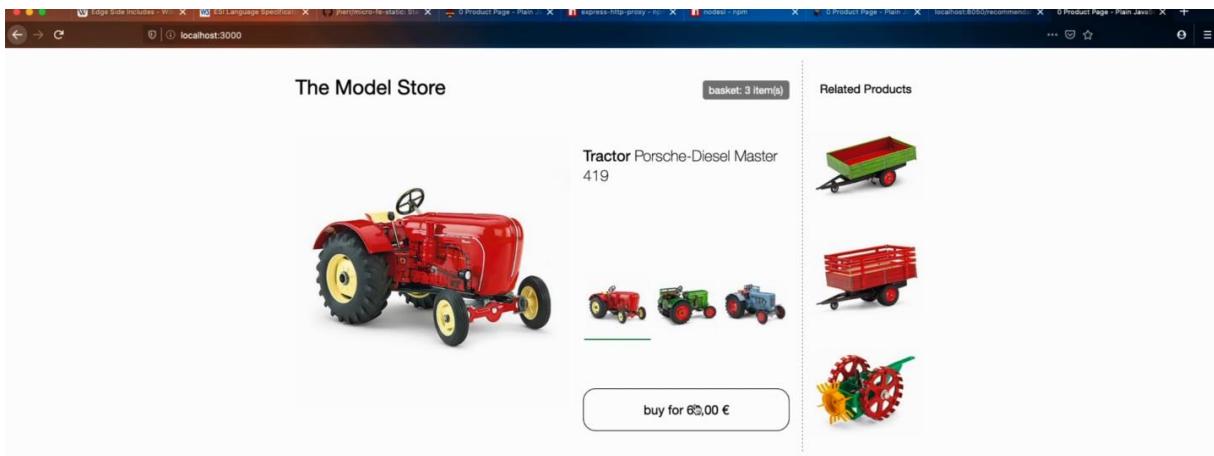
```

1 <div id="basket" class="empty">basket: 0 item(s)</div>
2 <script>
3 window.addEventListener('updateCart', (evt) => {
4   document.getElementById('basket').innerText =
5   `basket: ${evt.detail.cartCount} item(s)`;
6 });
7 </script>

```

We then add code for the basket to look for that event and then update the cart count to the new value





We now can use edge side includes for injected HTML code that can interact by firing and listening to custom events on the window, we can also use Redux or RxJS instead of the window object.

