Webpack 5 Module Federation: A game-changer in JavaScript architecture

Module federation allows a JavaScript application to dynamically run code from another bundle/build, on client and server.

Zack Jackson Follow
Mar 2, 2020 · 8 min read ★

This is the JavaScript bundler equivalent of what Apollo did with GraphQL.

> *A scalable solution to sharing code between independent applications has never been convenient, and near impossible at scale. The closest we had was externals or DLLPlugin, forcing centralized dependency on a external file. It was a hassle to share code, the separate applications were not truly standalone and usually, a limited number of dependencies are shared. Moreover, sharing actual feature code or components between separately bundled applications is unfeasible,* ***unproductive****, and* unprofitable.

. . .

What is Module Federation?

It's a type of JavaScript architecture I invented and prototyped. Then with the help of my co-creator and the founder of Webpack — it was turned into one of the most exciting features in the Webpack 5 core (there's some cool stuff in there, and the new API is really powerful and clean).

Module federation allows a JavaScript application to dynamically load code from another application — in the process, sharing dependencies, if an application consuming a federated module does not have a dependency needed by the federated code — Webpack will download the missing dependency from that federated build origin.

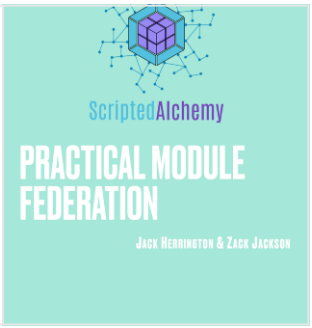
Code is shared if it can be, but fallbacks exist in each case. Federated code can always load its dependencies but will attempt to use the consumers' dependencies before downloading more payload. Less code duplication, dependency sharing just like a

monolithic Webpack build. While I may have invented this initial system — It was co-authored into Webpack 5 by myself (Zack Jackson) and Marais Rossouw with lots of guidance, pair-programming, and assistance from Tobias Koppers. These engineers played a key role in rewriting and stabilizing Module Federation within Webpack 5 core. Thank you for the continued collaboration and support.

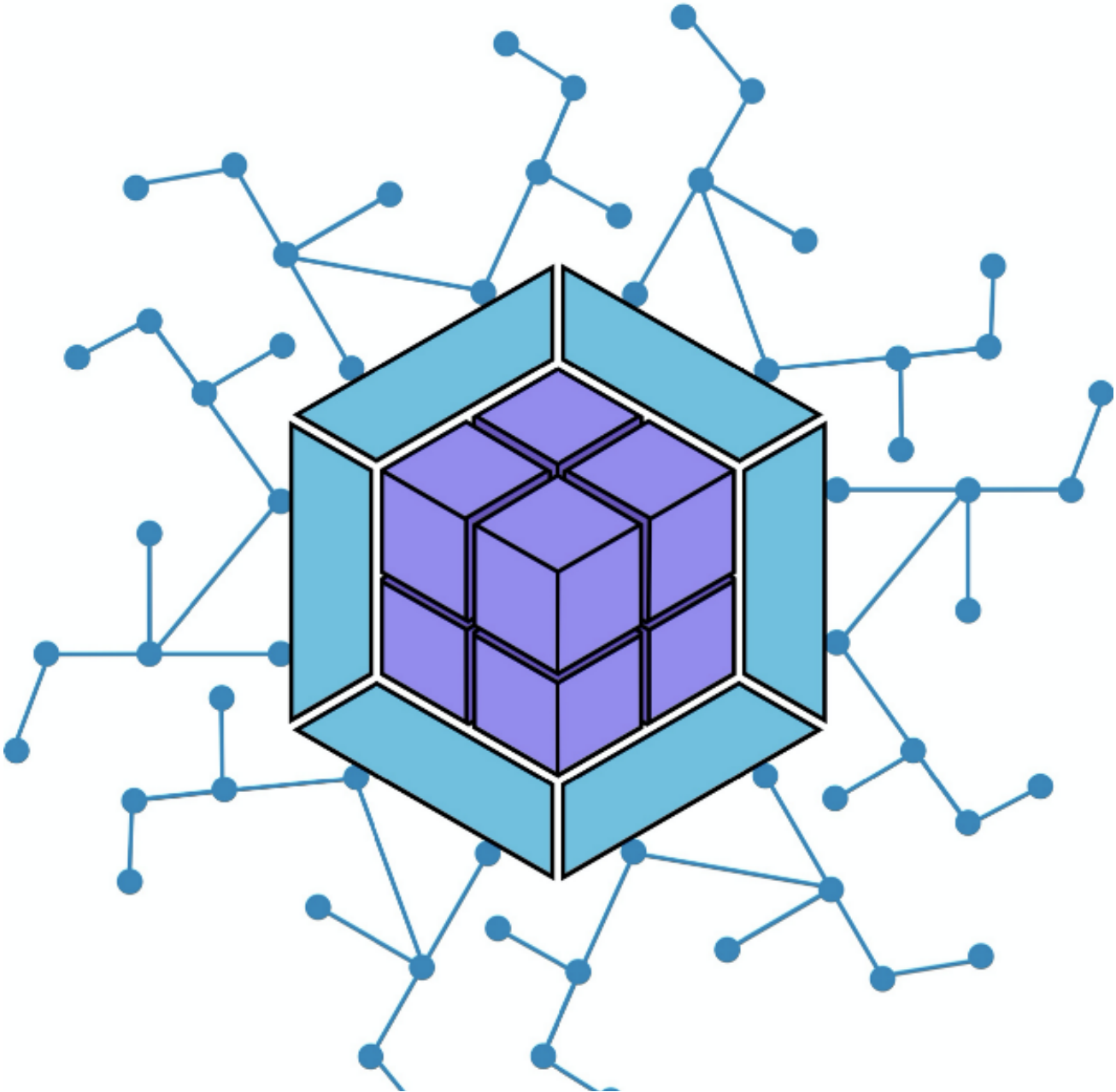
Practical Module Federation

"Practical Module Federation" is the first, and only, book on Webpack 5's innovative new live code sharing mechanism. It...

module-federation.myshopify.com



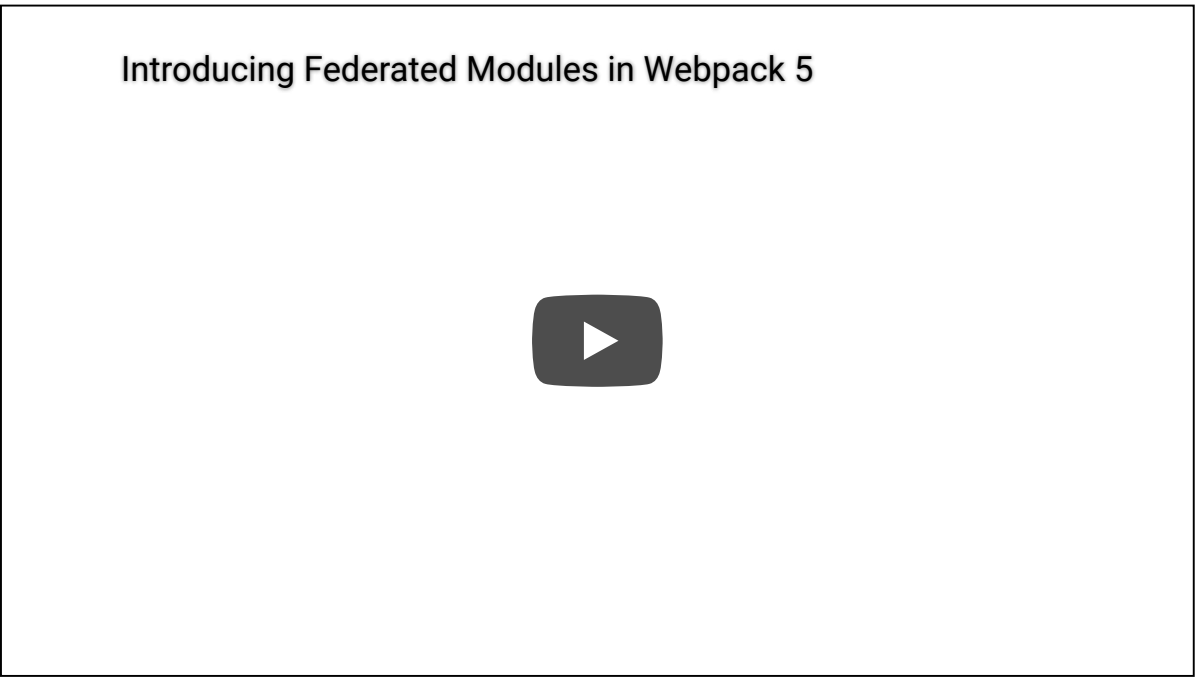
Terminology



- **Module federation**: same idea as Apollo GraphQL federation — but applied to JavaScript modules. In the browser and in node.js. Universal Module Federation
- **A host**: a Webpack build that is initialized first during a page load (when the onLoad event is triggered)
- **A remote**: another Webpack build, where part of it is being consumed by a "**host**"
- **Bidirectional-hosts**: when a bundle or Webpack build can work as a host or as a remote. Either consuming other applications or being consumed by others — at runtime
- Omnidirectional-hosts: hosts themselves don't know if they are host or remote on startup. This enables webpack to change out the hosts own vendors with ones based on semver rules. Allowing multiple versions when needed.

. . .

For those who want a more digestible version of this article, Jack Herrington made one!



> It's important to note that this system is designed so that each completely standalone build/app can be in its own repository, deployed independently, and run as its own independent SPA.

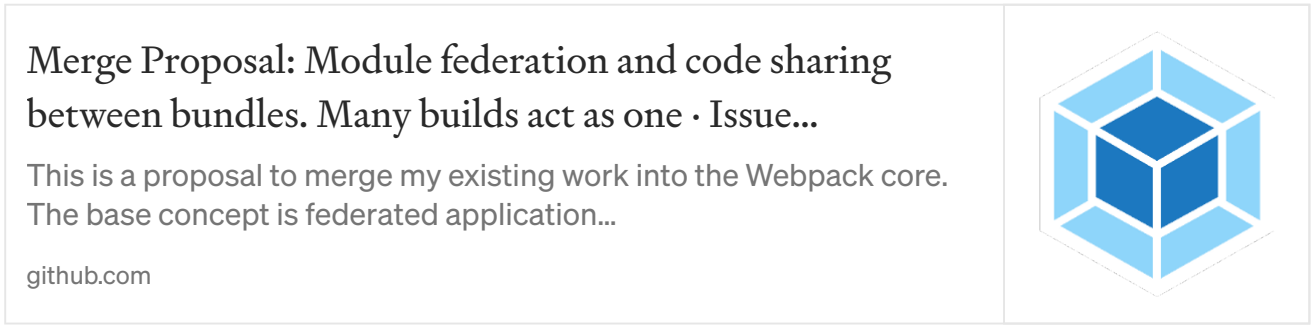
These applications are all **bi-directional hosts**. Any application **that's loaded first, becomes a host** — as you change routes and move through an application, loading federated modules in the same way you would implement dynamic imports. However if you were to refresh the page, whatever application first starts on that load, becomes a host.

> Let's say each page of a website is deployed and compiled independently. I want this micro-frontend style architecture but do not want and page reloads when changing routes, I also want to dynamically

> share code & vendors between them so it's just as efficient as if it was one large Webpack build, with code splitting.

Landing on the home page app would make the "home" page the "host", if you browse to an "about" page, the host (home page spa) is actually dynamically importing a module from another independent application (the about page spa). It doesn't load the main entry point and another entire application. **Only a few kilobytes of code.** If I am on the "about" page and refresh the browser. The "about" page becomes the "host" and browsing back to the home page again would be a case of the about page "host" Fetching a fragment of runtime from a "remote" — the home page. All applications are both remote and host, consumable and consumers of any other federated module in the system.

Read more on the technical aspects on GitHub:



Building a federated application

Let us start out with three standalone applications.

App One

Configuration:

I'm going to use the app container `<App>` from **App One.** It will be consumed by other applications. To do so, I `expose` Its `App` as `AppContainer`

App One will also consume components from **two other** federated applications. To do so, I specify the `remotes` scope

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const ModuleFederationPlugin =
require("webpack/lib/container/ModuleFederationPlugin");

module.exports = {
  // other webpack configs...
```

```
  plugins: [
    new ModuleFederationPlugin({
      name: "app_one_remote",
      remotes: {
        app_two: "app_two_remote",
        app_three: "app_three_remote"
      },
      exposes: {
        './AppContainer':'./src/App'
      },
      shared: ["react", "react-dom","react-router-dom"]
    }),
    new HtmlWebpackPlugin({
      template: "./public/index.html",
      chunks: ["main"]
    })
  ]
}
```

Setting up build orchestration:

In the head of my applications, I load `app_one_remote.js` — this connects you to other Webpack runtimes and provisions the orchestration layer at runtime, its a specially designed Webpack runtime and entry point. **Its not a normal application entry point and is only a few KB**

It's important to note these are special entry points — they are only a few KB in size. Containing a special Webpack runtime that can interface with the host, it is NOT a standard entry point

```
<head>
  <script src="http://localhost:3002/app_one_remote.js"></script>
  <script src="http://localhost:3003/app_two_remote.js"></script>
</head>
<body>
  <div id="root"></div>
</body>
```

Consuming code from a remote

App One has a page that consumes a dialog component from **App Two**

```
const Dialog = React.lazy(() => import("app_two_remote/Dialog"));

const Page1 = () => {
    return (
        <div>
            <h1>Page 1</h1>
            <React.Suspense fallback="Loading Material UI Dialog...">
```

```
                <Dialog />
            </React.Suspense>
        </div>
    );
}

export default Page1;
```

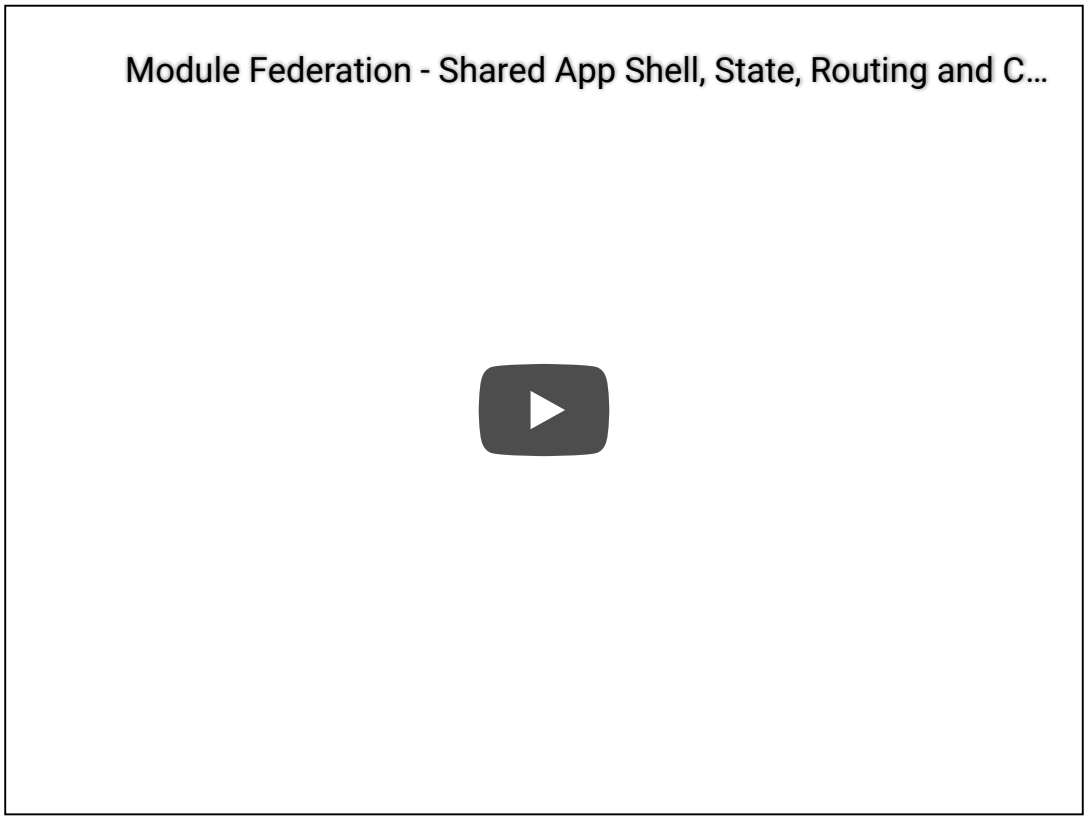
And the router looks pretty standard:

```
import { Route, Switch } from "react-router-dom";

import Page1 from "./pages/page1";
import Page2 from "./pages/page2";
import React from "react";

const Routes = () => (
  <Switch>
    <Route path="/page1">
      <Page1 />
    </Route>
    <Route path="/page2">
      <Page2 />
    </Route>
  </Switch>
);

export default Routes;
```



Configuration:

App Two will expose Dialog, enabling **App One** to consume it. App Two will also consume App One's `<App>` — so we specify `app_one` as a remote. Showcasing bi-directional hosts

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
const ModuleFederationPlugin =
require("webpack/lib/container/ModuleFederationPlugin");
module.exports = {
  plugins: [
    new ModuleFederationPlugin({
      name: "app_two_remote",
      library: { type: "var", name: "app_two_remote" },
      filename: "remoteEntry.js",
      exposes: {
        “./Dialog”: "./src/Dialog"
      },
      remotes: {
        app_one: "app_one_remote",
      },
      shared: ["react", "react-dom","react-router-dom"]
    }),
    new HtmlWebpackPlugin({
      template: "./public/index.html",
      chunks: ["main"]
    })
  ]
};
```

Consumption:

Heres what the root App looks like:

```
import React from "react";
import Routes from './Routes'
const AppContainer = React.lazy(() =>
import("app_one_remote/AppContainer"));

const App = () => {
    return (
        <div>
            <React.Suspense fallback="Loading App Container from
Host">
                <AppContainer routes={Routes}/>
            </React.Suspense>
        </div>
    );
```

```
}

export default App;
```

Heres what the default page looks like, that uses Dialog.

```
import React from 'react'
import {ThemeProvider} from "@material-ui/core";
import {theme} from "./theme";
import Dialog from "./Dialog";


function MainPage() {
    return (
        <ThemeProvider theme={theme}>
            <div>
                <h1>Material UI App</h1>
                <Dialog />
            </div>
        </ThemeProvider>
    );
}

export default MainPage
```

App Three

As expected, **App Three** looks similar. However, it does not consume the `<App>` from **App One** working more as a standalone, self-running component (no navigation or sidebar). As such, it does not specify any remotes

```
new ModuleFederationPlugin({
  name: "app_three_remote",
  library: { type: "var", name: "app_three_remote" },
  filename: "remoteEntry.js",
  exposes: {
    “./Button”: "./src/Button"
  },
  shared: ["react", "react-dom"]
}),
```

The end result in the browser (different to the first video)

Pay close attention to the network tab. The code is being federated across three different servers. Three different bundles. In general, id recommends against federating the whole application container unless you are taking advantage of SSR or progressive loading. The concept, however, is extraordinarily powerful.

Code Duplication

There is little to no dependency duplication. Through the `shared` option — **remotes will depend on host dependencies, if the host does not have a dependency, the remote will download its own. No code duplication, but built-in redundancy.**

While manually adding vendors or other modules to `shared` is not ideal at scale. This can be easily automated with a custom-written function, or with a supplemental Webpack plugin. We do plan to release `AutomaticModuleFederationPlugin` and maintain

it from outside of the Webpack Core. Now that we have built first-class code federation support into Webpack, extending its capabilities is trivial.

Now for the big question… **Does any of this work with SSR??**

It absolutely does

Server-Side Rendering

We have designed this to be ***Universal*** Module Federation works in any environment. Server-side rendering federated code is completely possible. Just have server builds use a commonjs library target. There are various ways to achieve federated SSR. S3 Streaming, ESI, automate an npm publish to consume server variants. I plan to use a commonly shared file volume or async S3 streaming to stream files across the filesystem. Enabling the server to require federated code just like it happens in the browser. Using `fs` instead of `http` to load federated code.

```
module.exports = {
 plugins: [
  new ModuleFederationPlugin({
   name: "container",
   library: { type: "commonjs-module" },
   filename: "container.js",
   remotes: {
    “./containerB”: "../1-container-full/container.js"
   },
   shared: ["react"]
  })
 ]
};
```

> “Module Federation does also work with `target: "node"`. Instead of URLs pointing to the other micro-frontends, here file paths are used which point to the other micro-frontends. That way you can do SSR with the same codebase and a different webpack config for building for node.js. The same properties stay true for Module Federation in node.js: e. g. Separate builds, Separate deploys” — Tobias Koppers

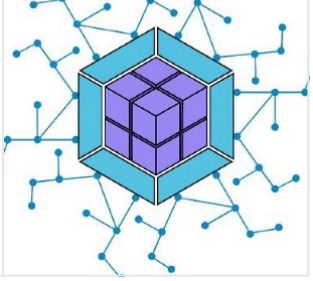
Federated Next.js on Webpack 5

Federation requires Webpack 5—which Next doesn’t officially support. However… i did manage to fork and upgrade Next.js to work with Webpack 5! It’s still a work in progress, some development mode middleware needs finishing touches. Production mode is working, some additional loaders still need to be re-tested.

feat: Upgrade Next.js to Webpack 5 by ScriptedAlchemy · Pull Request #2 · module-federation/next.js

Enabling module federation & suspense SSR

github.com



. . .

Talks, podcasts, or feedback

I'd love the opportunity to share more about this technology. If you want to use Module Federation or Federated architecture, we would hear about your experience And changes with current architecture. We also would love opportunities to speak about it on Podcasts, meetups, or corporations. Reach me on Twitter

Zack Jackson

The latest Tweets from Zack Jackson (@ScriptedAlchemy). Principal Engineer @lululemon. Distributed JavaScript...

twitter.com

You can also get hold of my co-creator, follow us for the latest updated on Module Federation, FOSA (Federation of Standalone Applications) Architecture, and other tools

we are creating — designed to work with Federated Applications

> Marais
>
> The latest Tweets from Marais (@codervandal). Frontend guru @ https://t.co/WRSY8p9lWc, co-creator of frontend...
>
> twitter.com

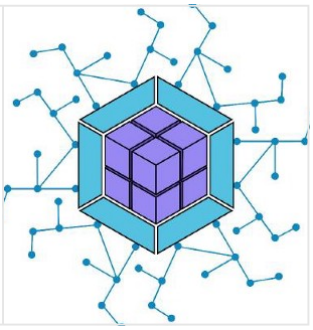
Examples of Module Federation

The community has had an enthusiastic response! My co-creators time, along with my own has been heavily focused on writing this into Webpack 5. We hope some code samples will help while we finalize some remaining features and write some documentation.

> Webpack 5 and Module Federation
>
> Picture this; you've got yourself a pretty whiz-bang component, not just any component, but that classic component that...
>
> dev.to

As we have the bandwidth, we will create SSR examples and more comprehensive demos. If anyone wants to build something that can be used as a demo — we gladly will accept pull requests back to `webpack-external-import`

> module-federation/module-federation-examples
>
> Examples showcasing Webpack 5's Module Federation. Contribute to module-federation/module-federation-examples...
>
> github.com



> ScriptedAlchemy/mfe-webpack-demo
>
> This Demonstrates Micro-frontend architecture using the FOSA application architecture specification. FOSA = Federation...
>
> github.com

ScriptedAlchemy/webpack-external-import

This project has been proposed for implementation into the Webpack core (with some rewrites and refactors). Track the...

github.com

Thanks to Tobias Koppers.

Sign up for Top 10 Stories

By The Startup

Get smarter at building your thing. Subscribe to receive The Startup's top 10 most read stories — delivered straight into your inbox, once a week. Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

JavaScript Webpack Web Development React Programming

Medium

About Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play