@module-federation/nextjs-mf

8.5.5 • Public • Published 9 days ago

Module Federation For Next.js

This plugin enables Module Federation on Next.js

Supports

- next ^14 || ^13 || ^12
- SSR included!

I highly recommend referencing this application which takes advantage of the best capabilities: https://github.com/module-federation/module-federation-examples

This project supports federated SSR

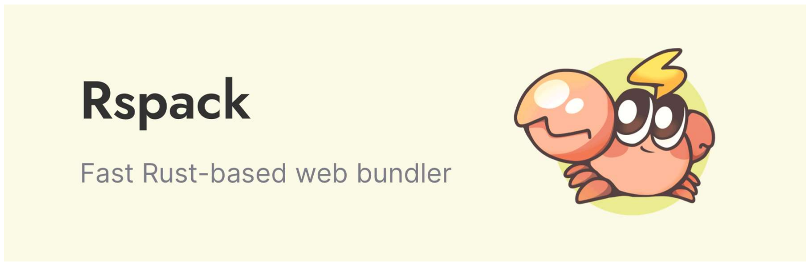
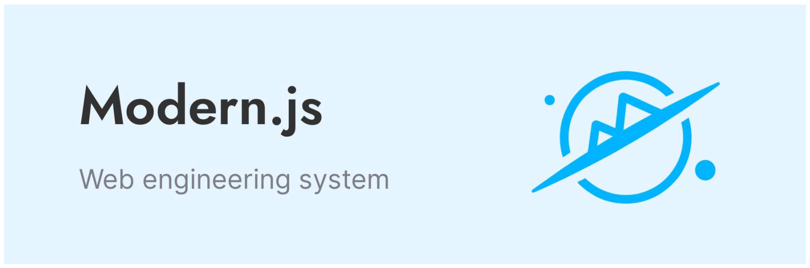
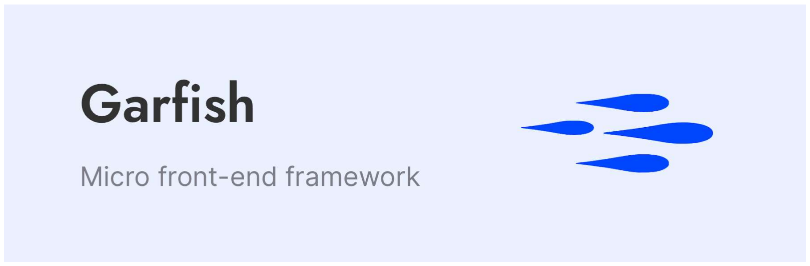
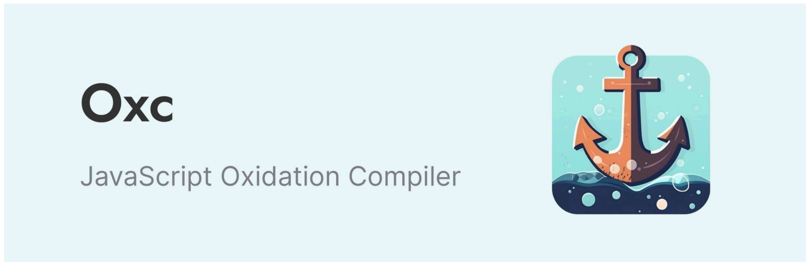
We are building a micro-frontend ecosystem!

While NextFederationPlugin "works", Next.js is staunchly opposed to the technology and Next is very difficult to support.

This plugin attempts to make the experience as seamless as possible, but it is not perfect.

We are building a micro-frontend ecosystem that is much more powerful than Next.js, built to support micro-frontends from the ground up.

If Federation is a big enabler for your teams and projects, please consider using our ecosystem thats designed for it

Rspack	
Modern.js	
Garfish	
Oxc	

Whats shared by default?

Under the hood we share some next internals automatically You do not need to share these packages, sharing next internals yourself will cause errors.

▶ See DEFAULT_SHARE_SCOPE:

Requirement

I set `process.env.NEXT_PRIVATE_LOCAL_WEBPACK = 'true'` inside this plugin, but its best if its set in env or command line export.

"Local Webpack" means you must have webpack installed as a dependency, and next will not use its bundled copy of webpack which cannot be used as i need access to all of webpack internals

- `NEXT_PRIVATE_LOCAL_WEBPACK=true next dev or next build`
- `npm install webpack`

Usage

```
import React, { lazy } from 'react';
const SampleComponent = lazy(() => import('next2/sampleComponent'));
```

To avoid hydration errors, use `React.lazy` instead of `next/dynamic` for lazy loading federated components.

See the implementation here: https://github.com/module-federation/module-federation-examples/tree/master/nextjs-v13/home/pages

With async boundary installed at the page level. You can then do the following

```
const SomeHook = require('next2/someHook');
import SomeComponent from 'next2/someComponent';
```

Demo

You can see it in action here: **https://github.com/module-federation/module-federation-examples/tree/master/nextjs-ssr**

Options

This plugin works exactly like ModuleFederationPlugin, use it as you'd normally. Note that we already share react and next stuff for you automatically.

Also NextFederationPlugin has own optional argument `extraOptions` where you can unlock additional features of this plugin:

```
new NextFederationPlugin({
  name: '',
  filename: '',
  remotes: {},
  exposes: {},
  shared: {},
  extraOptions: {
    debug: boolean, // `false` by default
    exposePages: boolean, // `false` by default
  },
});
```

- `debug` – enables debug mode. It will print additional information about what is going on under the hood.
- `exposePages` – exposes automatically all nextjs pages for you and theirs `./pages-map` .

Demo

You can see it in action here: **https://github.com/module-federation/module-federation-examples/pull/2147**

Implementing the Plugin

1. Use `NextFederationPlugin` in your `next.config.js` of the app that you wish to expose modules from. We'll call this "next2".

```
// next.config.js
// either from default
const NextFederationPlugin = require('@module-federation/nextjs-mf');

module.exports = {
  webpack(config, options) {
    const { isServer } = options;
    config.plugins.push(
      new NextFederationPlugin({
        name: 'next2',
        remotes: {
          next1: `next1@http://localhost:3001/_next/static/${isServer
        },
        filename: 'static/chunks/remoteEntry.js',
        exposes: {
          './title': './components/exposedTitle.js',
          './checkout': './pages/checkout',
        },
        shared: {
          // whatever else
        },
      }),
    );

    return config;
  },
};
```

```
// next.config.js

const NextFederationPlugin = require('@module-federation/nextjs-mf');

module.exports = {
  webpack(config, options) {
    const { isServer } = options;
    config.plugins.push(
      new NextFederationPlugin({
        name: 'next1',
        remotes: {
          next2: `next2@http://localhost:3000/_next/static/${isServer
        },
      }),
    );

    return config;
  },
};
```

4. Use react.lazy, low level api, or require/import from to import remotes.

```
import React, { lazy } from 'react';

const SampleComponent = lazy(() =>
  window.next2.get('./sampleComponent').then((factory) => {
    return { default: factory() };
  }),
);

// or
```

```
const SampleComponent = lazy(() => import('next2/sampleComponent'));

//or

import Sample from 'next2/sampleComponent';
```

RuntimePlugins

To provide extensibility and "middleware" for federation, you can refer to `@module-federation/runtime`

```
// next.config.js
new NextFederationPlugin({
  runtimePlugins: [require.resolve('./path/to/myRuntimePlugin.js')],
});
```

Utilities

Ive added a util for dynamic chunk loading, in the event you need to load remote containers dynamically.

```
import { loadRemote, init } from '@module-federation/runtime';
// if i have remotes in my federation plugin, i can pass the name of t
loadRemote('home/exposedModule')
// if i want to load a custom remote not known at build time.
init({
  name: 'hostname',
  remotes: [
    {
      name: 'home',
      entry: 'http://somthing.com/remoteEntry.js'
    }
  ],
    force: true // may be needed to sideload remotes after the fact.
  })
  loadRemote('home/exposedModule')
```

revalidate

Hot Reloading with `revalidate` in Production Environments

In production environments, ensuring that your server can dynamically reload and update without requiring a full restart is crucial for maintaining uptime and providing the latest features to your users without disruption. The `revalidate` utility from `@module-federation/nextjs-mf/utils` facilitates this by enabling hot reloading of the node server (not the client). This section outlines two implementations for integrating `revalidate` into your Next.js application to leverage hot reloading capabilities.

Preferred Implementation: Blocking Updates Before Rendering

This implementation is recommended for most use cases as it helps avoid hydration errors by ensuring that the server and client are always in sync. By blocking and checking for updates before rendering, you can guarantee that your application is always up-to-date without negatively impacting the user experience.

How it Works:

- **Before rendering the page**, the server checks if there are any updates available.
- **If updates are available**, it proceeds with Hot Module Replacement (HMR) before responding to the client request.
- **This method ensures** that all users receive the latest version of the application without encountering inconsistencies between the server-rendered and client-rendered content.

Implementation Example:

```
// __document.js

import { revalidate } from '@module-federation/nextjs-mf/utils';
import Document, { Html, Head, Main, NextScript } from 'next/document'
```

```
class MyDocument extends Document {
  static async getInitialProps(ctx) {
    if (ctx?.pathname && !ctx?.pathname?.endsWith('_error')) {
      await revalidate().then((shouldUpdate) => {
        if (shouldUpdate) {
          console.log('Hot Module Replacement (HMR) activated', shoulc
        }
      });
    }

    const initialProps = await Document.getInitialProps(ctx);
    return initialProps;
  }

  render() {
    return (
      <Html>
        <Head />
        <body>
          <Main />
          <NextScript />
        </body>
      </Html>
    );
  }
}
```

Stale Method: Post-Response Update Checks

While not recommended due to the potential for hydration errors, this method involves listening for the 'finish' event on the response object and then checking for updates. This could be useful in specific scenarios where updates can be applied less frequently or where immediate consistency between server and client is not as critical.

How it Works:

- **After responding to the client**, the server listens for the 'finish' event on the response object.
- **Once the response has been sent**, it checks for updates.
- **If updates are found**, it logs or acts upon these updates, although the updates will only apply to subsequent requests.

Implementation Example:

```
// Included in the `getInitialProps` method as shown in the preferred
ctx?.res?.on('finish', () => {
  revalidate().then((shouldUpdate) => {
    console.log('Response sent, checking for updates:', shouldUpdate);
  });
});
```

For Express.js

Hot reloading Express.js required additional steps: https://github.com/module-federation/core/blob/main/packages/node/README.md

Contact

If you have any questions or need to report a bug Reach me on Twitter @ScriptedAlchemy

Or join this discussion thread: https://github.com/module-federation/module-federation-examples/discussions/978

Keywords

none

Install

```
npm i @module-federation/nextjs-mf
```

Repository

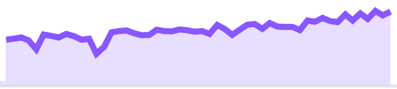
github.com/module-federation/core/tree/main

Homepage

github.com/module-federation/core/tree/main#readme

Weekly Downloads

66,530



Version

8.5.5

License

MIT

Unpacked Size

178 kB

Total Files

91

Issues

43

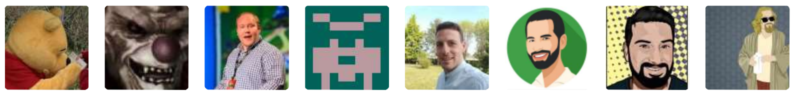
Pull Requests

28

Last publish

16 hours ago

Collaborators



>_Try on RunKit

Report malware

Support

Help

Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy