module-federation / module-federation-examples

Code Issues 34 Pull requests 19 Discussions Actions Projects

module-federation-examples / nextjs-dynamic-ssr /

ilteoood and ScriptedAlchemy chore: upgrade to storybook v8 (#4246)

49052f5 · last month

Name	Name	Last commit date
..		
checkout	fix(deps): update dependenc...	2 months ago
e2e	Remove and angular (#4087)	3 months ago
home	fix(deps): update dependenc...	2 months ago
shop	fix(deps): update dependenc...	2 months ago
README.md	Remove and angular (#4087)	3 months ago
cypress.env.json	Remove and angular (#4087)	3 months ago
index.spec.js	feat: nextjs dynamic ssr sam...	6 months ago
package.json	locks	2 months ago
pnpm-lock.yaml	chore: upgrade to storybook...	last month
pnpm-workspace.yaml	Remove and angular (#4087)	3 months ago

README.md

Next.js with Module Federation

Getting Started

2. run `pnpm run start` and browse to `http://localhost:3001` or one of the others

We are available to consult

Contact me zackary.l.jackson@gmail.com or @ScriptedAlchemy on Twitter

How it works?!

This implementaion leverages our propriatery *Software Streams* which allow me to stream commonjs modules at runtime to consuming apps. We have never made software streaming avaliable to the general public, while we have used it for 2 years and run several backends off the technology - its remained a heavily guarded secret. Software Streams is how SSR works, on the client side we are using enhanced federation interfaces to ensure that the top-level api works as expected.

It should allow `import()` , `require` , `import from` to work - this has been tested serverside but i have not yet tested anything else other than import() on the client.

There has been a leaked copy of an alpha from a year and a half ago for software streams. While it does work, there are several security flaws. The federation group has spend signigicant amounts of time enhancing streaming.

In the future, when this plugin is out of beta - we are planning to build in stream encryption to ensure that code streamed has not been manipulated in any way. This would rely on a salted cypher key that consumer and remote would know at build time.

We are also looking into running streamed software in a WASM isolate that cannot perform any damage, has no access to host resources. This would make it possible to execute untrusted code.

For the time being - I strongly suggest only federating trusted software between servers.

Security

In order to make this plugin work right out the box, the commonjs modules are exposed via `_next/static/ssr*` i strongly suggest having a CDN or piece of middleware that only allows access to this path from internal network or VPN. You do not want the public internet to be able to reach that path. You are exposing server code, where `process.browser` is not applied to tree shake server secrets since this is server code.

Context

We have three next.js applications

- `checkout` - port 3000
- `home` - port 3001
- `shop` - port 3002

The applications utilize omnidirectional routing and pages or components are able to be federated between applications like a SPA

I am using hooks here to ensure multiple copies of react are not loaded into scope on server or client.

The omnidirectional routing now hooks into webpack federation loading functions, so

load remotes when theres a know static import like `home/title`

I am using hooks here to ensure multiple copies of react are not loaded into scope on server or client.

Sharing

Next.js has all its internal modules pre-shared vis `@module-federation/nextjs-mf` you do need to share react via the plugin in order to ensure that the share scope runtime requirements are included - since you cannot share modules in a normal manner, like nextjs internls, the pre-shared modules are attached at runtime to the share scope. Any exta code you share is processed via the plugin which reconfigures sharing properly to work with next.js limitations.

The sharing limit is due to next not having any async boundary, theres no way to "pause" the application while webpack orchestrates share scope.

I am investigating new methods that may solve the module sharing problem in next.js, however this is a complex problem to solve and requires enormus amounts of knowladge around how webpack and federation work inside the module graph.

Running Cypress E2E Tests

To run tests in interactive mode, run `npm run cypress:debug` from the root directory of the project. It will open Cypress Test Runner and allow to run tests in interactive mode. More info about "How to run tests"

To build app and run test in headless mode, run `yarn e2e:ci`. It will build app and run tests for this workspace in headless mode. If tets failed cypress will create `cypress` directory in sample root folder with screenshots and videos.

"Best Practices, Rules amd more interesting information here