npm.io

Search package

0.7.1 • Published 28 days ago

@inertiajs/inertia-react v0.7.1

Weekly downloads	License
2,039	MIT
Repository	Last release
github	28 days ago

Share package

Inertia.js React Adapter

Installation

Install using NPM:

```
npm install @inertiajs/inertia @inertiajs/inertia-react --save
```

Configure server-side framework

The first step when using Inertia.js is to configure your server-side framework. This primarily includes setting up a root template and

updating your endpoints to return a proper Inertia response. For an example of this, see our Laravel adapter.

Setting up Webpack

Here is an example Webpack configuration that uses Laravel Mix. Note the `@` alias to the `/resources/js` directory.

```
const mix = require('laravel-mix')
const path = require('path')

mix
  .react('resources/js/app.js', 'public/js')
  .sass('resources/sass/app.scss', 'public/css')
  .webpackConfig({
    output: { chunkFilename: 'js/[name].js?id=[chunkhash]' },
    resolve: {
      alias: {
        '@': path.resolve('resources/js'),
      },
    },
  })
```

Setup dynamic imports

We recommend using code splitting with Inertia.js. To do this we need to enable dynamic imports. We'll use a Babel plugin to make this work. First, install the plugin:

```
npm install @babel/plugin-syntax-dynamic-import --save
```

Next, create a `.babelrc` file in your project with the following:

```
{
  "plugins": ["@babel/plugin-syntax-dynamic-import"]
}
```

Alternatively, if you're using Laravel Mix, you can put this in your `webpack.mix.js` file:

```
mix.babelConfig({
  plugins: ['@babel/plugin-syntax-dynamic-import'],
})
```

Initializing React

Next, update your main JavaScript file to boot your Inertia app. All we're doing here is initializing React with the base Inertia page component.

```
import { InertiaApp } from '@inertiajs/inertia-react'
import React from 'react'
import { render } from 'react-dom'

const app = document.getElementById('app')

render(
  <InertiaApp
    initialPage={JSON.parse(app.dataset.page)}
    resolveComponent={name => import(`@/Pages/${name}`).then(module
=> module.default)}
  />,
  app
)
```

The `resolveComponent` is a callback that tells Inertia how to load a page component. It receives a page name (string), and must return a component instance.

Using Inertia without code splitting

It's possible to also use Inertia without code splitting. This will generate one larger JavaScript bundle, instead of many smaller ones. With this approach, the dynamic imports Babel plugin is not required.

One way to do this is manually loading all your page components:

```
import { InertiaApp } from '@inertiajs/inertia-react'
import React from 'react'
import { render } from 'react-dom'
```

```
const app = document.getElementById('app')

const pages = {
  'Dashboard/Index': require('./Pages/Dashboard/Index').default,
  'Users/Index': require('./Pages/Users/Index').default,
  'Users/Create': require('./Pages/Users/Create').default,
  // etc...
}

render(
  <InertiaApp
    initialPage={JSON.parse(app.dataset.page)}
    resolveComponent={name => pages[name]}
  />,
  app
)
```

Another option is to use `required.context` to automatically register all your page components.

```
import { InertiaApp } from '@inertiajs/inertia-react'
import React from 'react'
import { render } from 'react-dom'

const app = document.getElementById('app')

const files = require.context('./', true, /\.js$/i)

render(
  <InertiaApp
    initialPage={JSON.parse(app.dataset.page)}
    resolveComponent={page => files(`./Pages/${page}.js`).default}
  />,
  app
)
```

Creating a base layout

While not required, for most projects it makes sense to create a default site layout that your specific pages can extend. Save this to `/Shared/Layout.js`.

```
import { InertiaLink } from '@inertiajs/inertia-react'
import React from 'react'
```

```
export default function Layout({ children }) {
  return (
    <main>
      <header>
        <InertiaLink href="/">Home</InertiaLink>
        <InertiaLink href="/about">About</InertiaLink>
        <InertiaLink href="/contact">Contact</InertiaLink>
      </header>

      <article>{children}</article>
    </main>
  )
}
```

Creating page components

With Inertia.js, each page in your application is a JavaScript component. Here's an example of a page component. Save this to `/Pages/Welcome.js`. Note how it extends the `Layout.js` component we created above.

```
import React from 'react'
import Layout from '@/Shared/Layout'

export default function Welcome() {
  return (
    <Layout>
      <h1>Welcome</h1>
      <p>Welcome to my first Inertia.js app!</p>
    </Layout>
  )
}
```

Creating links

To create an Inertia link, use the `<InertiaLink>` component.

```
import { InertiaLink } from '@inertiajs/inertia-react'
import React from 'react'

export default () => <InertiaLink href="/">Home</InertiaLink>
```

You can also specify the browser history and scroll behaviour. By default all link clicks "push" a new history state, and reset the scroll position back to the top of the page. However, you can override these defaults using the `replace` and `preserve-scroll` attributes.

```
<InertiaLink replace preserve-scroll href="/">Home</InertiaLink>
```

You can also specify the method for the request. The default is `GET`, but you can also use `POST`, `PUT`, `PATCH`, and `DELETE`.

```
<InertiaLink href="/logout" method="post">Logout</InertiaLink>
```

You can add data using the `data` attribute:

```
<InertiaLink href="/endpoint" method="post" data={{ foo:
bar}}>Save</InertiaLink>
```

You can also preserve a page component's local state using the `preserveState` attribute. This will prevent a page component from fully re-rendering. This is especially helpful with forms, since you can avoid manually repopulating input fields, and can also maintain a focused input.

```
<input onChange={this.handleChange} value={query} />
<InertiaLink href="/search" data={query}>Search</InertiaLink>
```

Manually making visits

In addition to clicking links, it's also very common to manually make Inertia visits. The following methods are available. Take note of the defaults.

```
import { Inertia } from '@inertiajs/inertia'

// Make a visit
Inertia.visit(url, { method: 'get', data: {}, replace: false,
preserveState: false, preserveScroll: false })

// Make a "replace" visit
```

```
Inertia.replace(url, { method: 'get', data: {}, preserveState: true,
preserveScroll: false })

// Make a "replace" visit to the current url
Inertia.reload({ method: 'get', data: {}, preserveState: false,
preserveScroll: false })

// Make a POST visit
Inertia.post(url, data, { replace: false, preserveState: true,
preserveScroll: false })

// Make a PUT visit
Inertia.put(url, data, { replace: false, preserveState: true,
preserveScroll: false })

// Make a PATCH visit
Inertia.patch(url, data, { replace: false, preserveState: true,
preserveScroll: false })

// Make a DELETE visit
Inertia.delete(url, { replace: false, preserveState: false,
preserveScroll: false })
```

Just like with an `<InertiaLink>`, you can control the history control behaviour using `replace`, scroll behaviour using `preserveScroll`, and local component state behaviour using `preserveState`.

Accessing page data in other components

Sometimes it's necessary to access the page data (props) from a non-page component. One really common use-case for this is the site layout. For example, maybe you want to show the currently authenticated user in your site header. This is possible using React's context feature. The base Inertia component automatically provides the current page via context, which can then be accessed by a consumer later on.

The easiest way to access page props is with our `usePage` hook.

```
import { InertiaLink, usePage } from '@inertiajs/inertia-react'
import React from 'react'

export default function Layout({ children }) {
  const { auth } = usePage()
```

```
  return (
    <main>
      <header>
        You are logged in as: {auth.user.name}

        <nav>
          <InertiaLink href="/">Home</InertiaLink>
          <InertiaLink href="/about">About</InertiaLink>
          <InertiaLink href="/contact">Contact</InertiaLink>
        </nav>
      </header>

      <article>{children}</article>
    </main>
  )
}
```

Remembering local component state

When navigating browser history, Inertia reloads pages using prop data cached in history state. Inertia does not, however, cache local component state, since this is beyond its reach. This can lead to outdated pages in your browser history. For example, if a user partially completes a form, then navigates away, and then returns back, the form will be reset and their work will have been lost.

To mitigate this issue, you can use the `useRememberedState` hook to tell Inertia.js which local component state to cache.

```
import { useRememberedState } from '@inertiajs/inertia-react'
import React from 'react'

export default function Profile() {
  const [formState, setFormState] = useRememberedState({
    first_name: null,
    last_name: null,
    // ...
  })

  // ...
}
```

If your page contains multiple components using the remember functionality, you'll need to provide a unique key for each component.

For example, `Users/Create`. If you have multiple instances of the same component on the page, be sure to include a unique identifier for each of those instances. For example, `Users/Edit:{id}`.

```
import { useRememberedState } from '@inertiajs/inertia-react'
import React from 'react'

export default function Profile() {
  const [formState, setFormState] = useRememberedState({
    first_name: props.user.first_name,
    last_name: props.user.last_name,
    // ...
  }, `Users/Edit:${props.user.id}`)

  // ...
}
```

1 dependency | 2 dependents | 39 versions

lodash.isequal

contact@npm.io